C16

```
128   *
129   * Parameters:
130   *   None.
131   *
132   * Returns:
133   *   The new Combo Box
134   *
135   ***********************************************************************/
136
137 static CBoxPtr REST_CalCreateDateBox (void)
138 1 {
139 1   CBoxPtr      newBox;  /* The new combo box created */
140 1   ColorPtr     fgColor; /* The color to set the foreground to */
141 1   ColorPtr     bgColor; /* The color to set the background to */
142 1   PenPtr       pen;     /* The pen to use for the combo box */
143 1   FontPtr      font;    /* The font to use for the combo box */
144
145 1   /*
146 1    * Create the combo box and set the defaults
147 1    */
148
149 1   newBox = CBOX_Create ();
150 1   CBOX_SetLabel (newBox, "");
151 1   CBOX_SetSubClass (newBox, CBOX_TYPEDROPDOWNLIST);
152
153 1   /* Set the resizing so that it doesn't resize at all */
154 1   WGT_SetRzFlags ((WgtPtr)newBox,
155 1                   WGT_RESIZEKEEPWIDTH | WGT_RESIZEKEEPHEIGHT |
156 1                   WGT_RESIZEKEEPLEFT | WGT_RESIZEKEEPRIGHT);
157
158 1   /*
159 1    * Attempt to get each 'eden' default resource and set it
160 1    */
161
162 1   if ((fgColor = (ColorPtr)RES_Find ("eden", "CBoxPgColor")) != NULL)
163 1     WGT_SetFgColor ((WgtPtr)newBox, fgColor);
164
165 1   if ((bgColor = (ColorPtr)RES_Find ("eden", "CBoxBgColor")) != NULL)
166 1     WGT_SetBgColor ((WgtPtr)newBox, bgColor);
167
168 1   if ((pen = (PenPtr)RES_Find ("eden", "CBoxPen")) != NULL)
169 1     WGT_SetPen ((WgtPtr)newBox, pen);
170
171 1   if ((font = (FontPtr)RES_Find ("eden", "CBoxFont")) != NULL)
172 1     WGT_SetFont ((WgtPtr)newBox, font);
173
174 1   /* Return the new box */
175 1   return (newBox);
176   }
```



```
178 /***********************************************************************
179  * REST_CalCreateDateText
180  *
181  * Description:
182  *   This routine will create a TEd to be used for displaying
183  *   a singl restocal time for a day.
184  *
185  * Parameters:
186  *   None.
187  *
188  * Returns:
189  *   The new Text Area
190  *
191  ***********************************************************************/
192
193 static TEdPtr REST_CalCreateDateText (void)
194 1 {
195 1   TEdPtr       newText;  /* The new text area created */
196 1   ColorPtr     fgColor;  /* The color to set the foreground to */
197 1   ColorPtr     bgColor;  /* The color to set the background to */
198 1   PenPtr       pen;      /* The pen to use for the combo box */
199 1   FontPtr      font;     /* The font to use for the text area */
200
201 1   /*
202 1    * Create the text area and set the defaults
203 1    */
204
205 1   newText = TED_Create ();
206 1   TED_SetJustif (newText, DRAW_JUSTCENTER);
207 1   TED_SetOptFlags (newText, TED_OPTOUTPUTONLY);
208
209 1   /* Set the resizing so that it doesn't resize at all */
210 1   WGT_SetRzFlags  ((WgtPtr)newText,
211 1                    WGT_RESIZEKEEPWIDTH | WGT_RESIZEKEEPHEIGHT |
212 1                    WGT_RESIZEKEEPLEFT | WGT_RESIZEKEEPRIGHT);
213
214 1   /*
215 1    * Attempt to get each 'eden' default resource and set it
216 1    */
217
218 1   if ((fgColor = (ColorPtr)RES_Find (
219 1                                      "eden", "OutputFgColor")) != NULL)
220 1     WGT_SetFgColor ((WgtPtr)newText, fgColor);
221 1   if ((bgColor = (ColorPtr)RES_Find (
222 1                                      "eden", "OutputBgColor")) != NULL)
223 1     WGT_SetBgColor ((WgtPtr)newText, bgColor);
224 1   if ((pen = (PenPtr)RES_Find ("eden", "OutputPen")) != NULL)
225 1     WGT_SetPen ((WgtPtr)newText, pen);
226
227 1   if ((font = (FontPtr)RES_Find ("eden", "OutputFont")) != NULL)
228 1     WGT_SetFont ((WgtPtr)newText, font);
229
230 1   /* Return the new text area */
231 1   return (newText);
232   }
```

```
234  /*****************************************************************
235   * REST_CalSetBoxPos
236   *
237   * Description:
238   *   This routine will set the position for the combo box and TEd
239   *   for the given day given the coordinates for the day's entire box.
240   *
241   * Parameters:
242   *   dayNumber (I) - The day to update the combo box and TEd for.
243   *   dayOri    (I) - The origin coordinates for the day's box.
244   *   dayExt    (I) - The extent coordinates for the day's box.
245   *
246   * Returns:
247   *   None.
248   *
249   *****************************************************************/
250
251  static void REST_CalSetBoxPos (Int         dayNumber,
252                                 Point16Rec dayOri,
253                                 Point16Rec dayExt)
254  1 {
255  1   Rect16Rec    box;  /* Bounding box for the combo box and TEd */
256  1
257  1   /* Set the width to leave an offset on each side */
258  1   box.Ext.x = dayExt.x - (2 * DAY_LABEL_OFFSET);
259  1
260  1   /* If this is too wide use the max width */
261  1   if (box.Ext.x > DATE_CBOX_WIDTH)
262  2   {
263  2     box.Ext.x = DATE_CBOX_WIDTH;
264  1   }
265  1
266  1   /* Center the box, horizontally */
267  1   box.Ori.x = dayOri.x + ((dayExt.x - box.Ext.x) / 2);
268  1
269  1   /* Set the height of the box */
270  1   box.Ext.y = DATE_CBOX_HEIGHT;
271  1
272  1   /* Center the box, vertically */
273  1   box.Ori.y = dayOri.y + ((dayExt.y - DATE_TEXT_HEIGHT ) / 2);
274  1
275  1   /* Set the combo box's bounding box */
276  1   WGT_SetBox ((WgtPtr)backupTimeBox[dayNumber], &box);
277  1
278  1   /* Set the text area's bounding box */
279  1   box.Ext.y = DATE_TEXT_HEIGHT;
280  1   WGT_SetBox ((WgtPtr)backupTimeText[dayNumber], &box);
281
282  }
```



```
283  /*****************************************************************
284   * REST_GetDayWidthHeight
285   *
286   * Description:
287   *   This routine will determine the current width and height of a day
288   *   in the calendar window.
289   *
290   * Parameters:
291   *   width (O)  - The width of the days.
292   *   height (O) - The height of the days.
293   *
294   * Returns:
295   *   None.
296   *
297   *****************************************************************/
298
299  void REST_GetDayWidthHeight (Int  *width,
300                               Int  *height)
301  1 {
302  1   int          numberDays;
303  1                                /* Number of days in the displayed month */
304  1   int          numberWeeks;
305  1                                /* Number of weeks in the displayed month */
306  1   Point16Ptr   panelExt;    /* Extents of the panel */
307  1   Int          panelWidth;  /* The drawable panel width */
308  1   Int          panelHeight; /* The drawable panel height */
309  1
310  1   /* Determine the number of days and weeks in the displayed month */
311  1   numberDays = GTIME_DaysPerMonth (displayedMonth, displayedYear);
312  1   numberWeeks = GTIME_NumberWeeks (
313  1                                displayedMonthTime.tm_wday, numberDays);
314  1
315  1   /* Get the current extents of the drawing panel */
316  1   panelExt = (Point16Ptr) WGT_GetExt ((
317  1                                WgtPtr)REST_CalendarWindow->CalendarPanel);
318  1   panelWidth = panelExt->x - (2 * MARGIN_WIDTH);
319  1   panelHeight = panelExt->y - 1;
320  1
321  1   /* Get the width and height of each day */
322  1   *width = panelWidth / DAYS_PER_WEEK;
323  1   *height = (panelHeight - DAY_LABEL_HEIGHT) / numberWeeks;
324
325  }
```

```
/*****************************************************************
 * REST_DrawSelectedDay
 *
 * Description:
 *   This routine will draw the border and text number for the
 *   selected day.
 *
 * Parameters:
 *   drawSelected - Flag if the day should still be drawn selected.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

static void REST_DrawSelectedDay (BoolEnum drawSelected)
{
  Rect16Rec     rect;         /* Rectangle to draw */
  int           dayNumber;    /* Day number (0-6) of the selected day */
  int           weekNumber;   /* Week number of the selected day */
  int           dayWidth;     /* The width of a day box */
  int           dayHeight;    /* The height of a day box */

  /* Only draw if there is a selected day */
  if (selectedDay >= 0)
  {

    /* Only draw if the selected day is in the current month/year */
    if ((selectedMonth == displayedMonth) &&
                                    selectedYear == displayedYear))

    {

      /* Get the width and height of the day */
      REST_GetDayWidthHeight (&dayWidth, &dayHeight);

      /* Determine the day number (0-6) */
      dayNumber = ((selectedDay % DAYS_PER_WEEK) +
                   displayedMonthTime.tm_wday) % DAYS_PER_WEEK;

      /* Determine the week number */
      weekNumber = (
            selectedDay + displayedMonthTime.tm_wday) / DAYS_PER_WEEK;

      /* Calculate the bounding box for the selected day */
      rect.Ori.x = MARGIN_WIDTH + dayWidth * dayNumber;
      rect.Ori.y = DAY_LABEL_HEIGHT + (dayHeight * weekNumber);
      rect.Ext.x = dayWidth;
      rect.Ext.y = dayHeight;

      /* If the day is still selected, draw the border */
      if (drawSelected)
      {
        /* Draw in one pixel all the way around,
                                                to preserve the border */
        rect.Ori.x++;
        rect.Ori.y++;
        rect.Ext.x--;
        rect.Ext.y--;

        /* Draw a thick border around the selected day */
        DRAW_SetPen((
              WgtPtr)REST_CalendarWindow->CalendarPanel, PEN_Solid2());
        DRAW_SetColors ((WgtPtr)REST_CalendarWindow->CalendarPanel,
                        COLOR_Black(),
```



```
                        COLOR_Transparent());
        DRAW_Rect ((WgtPtr)REST_CalendarWindow->CalendarPanel, &rect);

      }

      /* If not still selected, just invalidate the region */
      else
      {
        /* Redraw a large enough rectangle to cover the entire day */
        rect.Ori.x -= BORDER_OFFSET;
        rect.Ori.y -= BORDER_OFFSET;
        rect.Ext.x += (2 * BORDER_OFFSET);
        rect.Ext.y += (2 * BORDER_OFFSET);
        DRAW_InvalRect ((
                        WgtPtr)REST_CalendarWindow->CalendarPanel, &rect);
      }
    }
  }
}
```

```
 403  /*****************************************************************
 404   * REST_CalendarSelectDay
 405   *
 406   * Description:
 407   *    This routine will select the day at the location of the current
 408   *    mouse click.
 409   *
 410   * Parameters:
 411   *    None.
 412   *
 413   * Returns:
 414   *    BOOL_TRUE  - If the day was successfully selected
 415   *    BOOL_FALSE - otherwise
 416   *
 417   *****************************************************************/
 419  BoolEnum REST_CalendarSelectDay (void)
 420  {
 421      Point16Rec loc;                         /* Location of the mouse click */
 422      Rect16Ptr  wgtBox;
                                                 /* Bounding box for the combo box */
 423      Rect16Rec  box;
                                                 /* Box for the combo box with border */
 424      Point16Rec penSize;                     /* Size of the combo box's pen */
 425      BoolEnum   inTimeBox = BOOL_FALSE;
                                                 /* Flag if click was in the combo box */
 426      int        dayNumber;
                                                 /* Number of the day column (0-6) */
 427      int        weekNumber;                  /* Number of the week (row) */
 428      int        numberDays;
                                                 /* Number of days in the displayed month */
 429      int        boxNumber;                   /* Total box number */
 430      int        date;                        /* Day of the month */
 431      BoolEnum   returnStatus;
                                                 /* Flag if the click was in a valid date */
 432      Int        index;
                                                 /* Index for time on selected day */
 433      Int        dayWidth;                    /* The width of a day box */
 434      Int        dayHeight;                   /* The height of a day box */

 436      /* Get the location of the click */
 437      EVENT_QueryWgtLoc((WgtPtr)REST_CalendarWindow->CalendarPanel, &loc);

 439      /* Get the width and height of the day */
 440      REST_GetDayWidthHeight (&dayWidth, &dayHeight);

 442      /* Get the day (column) and week (row) */
 443      dayNumber  = ((loc.x - MARGIN_WIDTH)/ dayWidth) + 1;
 444      weekNumber = (loc.y - DAY_LABEL_HEIGHT) / dayHeight;

 446      /* Validate the location against the drawing area */
 447      if ((loc.y < DAY_LABEL_HEIGHT) ||
 448          (loc.x < MARGIN_WIDTH) ||
 449          (loc.x > (MARGIN_WIDTH + (dayWidth * DAYS_PER_WEEK))) ||
 450          (dayNumber > DAYS_PER_WEEK))
 451      {
 452        returnStatus = BOOL_FALSE;
 453      }
 454      else
 455      {
 456        /* Get the overall box number */
 457        boxNumber = dayNumber + (weekNumber * DAYS_PER_WEEK);

 459        /* Determine the date */
 460        date = boxNumber - displayedMonthTime.tm_wday;
```



```
 462      /* Validate the date,
                                        make sure at least one backup was done on that day */
 463      numberDays = GTIME_DaysPerMonth (displayedMonth, displayedYear);
 464      if ((date > 0) && (date <= numberDays) && (
                                                    backupTimesCount[date-1] > 0))
 465      {
 467        /*
 468         * Don't include clicks that are inside the combo box
 469         */

 471        /* If there is more than one backup on the date */
 472        if (backupTimesCount[date-1] > 1)
 473        {
 474          /* Get the sizes */
 475          PEN_QuerySize (WGT_GetPen (backupTimeBox[date - 1]), &penSize);
 476          wgtBox = (Rect16Ptr) WGT_GetBox (backupTimeBox[date - 1]);

 478          /* Get the overall area of the combo box */
 479          box.Ori.x = wgtBox->Ori.x - penSize.x;
 480          box.Ori.y = wgtBox->Ori.y - penSize.y;
 481          box.Ext.x = wgtBox->Ext.x + (4 * penSize.x);
 482          box.Ext.y = wgtBox->Ext.y + (4 * penSize.y);

 484          /* Determine if the click was in the box */
 485          inTimeBox = RECT_ContainsPoint(&box, &loc);
 486        }

 488        /* Continue if the click was not in a combo box */
 489        if (!inTimeBox)
 490        {

 492          /* Redraw the old selected day (has affect of unselecting) */
 493          REST_DrawSelectedDay (BOOL_FALSE);

 495          /* Select the date that was clicked on */
 496          selectedDay = date - 1;
 497          selectedMonth = displayedMonth;
 498          selectedYear = displayedYear;

 500          /* Set the currently selected time */
 501          if (backupTimesCount[date-1] == 1)
 502          {
 503            /* Obvious choice, there's only one */
 504            currentSelectedTime = backupTimes[selectedDay][0];
 505          }
 506          else
 507          {
 508            /* Get the currently chosen index (
                                                    that's what the user sees) */
 509            index = (Int) CBOX_ChosenGetId (backupTimeBox[selectedDay]);
 510            currentSelectedTime = backupTimes[selectedDay][index];
 511          }

 513          /* Draw the newly selected day */
 514          REST_DrawSelectedDay (BOOL_TRUE);

 516          /* Flag that the selection was accepted */
 517          returnStatus = BOOL_TRUE;
 518        }
 519        else
 520        {
 521          /* Flag that the selection was rejected */
 522          returnStatus = BOOL_FALSE;
 523        }
```

```
524 2     }
525 2     else
526 1     {
527 3       /* Flag that the selection was rejected */
528 3       returnStatus = BOOL_FALSE;
529 2     }
530 1   }

532 1   /* Return the status */
533 1   return (returnStatus);
534   }
```



```
536   /*****************************************************************
537    * REST_BackupExistsInMonth
538    *
539    * Description:
540    *   This routine will determine if a backup exists in the given month
541    *   for the given year.
542    *
543    * Parameters:
544    *   month (I) - Index of month to check
545    *   year  (I) - Year
546    *
547    * Returns:
548    *   BOOL_TRUE  - If a backup was done in the given month/year
549    *   BOOL_FALSE - otherwise
550    *
551    *****************************************************************/

553   static BoolEnum REST_BackupExistsInMonth (Int      month,
554                                             Int      year)
555 1 {
556 1   int         startDay = 1;               /* First day of the month */
557 1   time_t      endMonthTime;
                                              /* Last time value for the month */
558 1   time_t      startMonthTime;
                                              /* First time value for the month */
559 1   struct tm   thisTime;                   /* Time for current query */
560 1   long        cookie = INIT_COOKIE;       /* Ahh, the magic cookie */
561 1   short       numberEntries;
                                              /* Number of entries returned */
562 1   time_t     *times;
                                              /* Array of valid backup times */
563 1   eerrno_ty   eerrno;                     /* Error code returned */
564 1   BoolEnum    backupsExist = BOOL_FALSE;/* Flag if backups exist */
565 1   u_long      flags;

567 1   if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
568 1     flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
569 1   else
570 1     flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

572 1   /* Update the month and year in case of bad range */
573 1   GTIME_UpdateDate (&startDay, &month, &year, BOOL_FALSE);

575 1   /* Get the start time of the month */
576 1   GTIME_GetTimeTm (startDay,
577 1                   month,
578 1                   year,
579 1                   0,
580 1                   0,
581 1                   &thisTime);
582 1   startMonthTime = mktime (&thisTime);

584 1   /* Get the End time of the month */
585 1   GTIME_GetTimeTm (GTIME_DaysPerMonth(month, year),
586 1                   month,
587 1                   year,
588 1                   HOURS_PER_DAY - 1,
589 1                   MINUTES_PER_HOUR - 1,
590 1                   &thisTime);
591 1   endMonthTime = mktime (&thisTime);

593 1   /* Attempt to get 1 time during the month */
594 1   times = (time_t *) GUTIL_Malloc (sizeof(time_t));
595 1   if ((eerrno = EDMRST_GetAllBackupTimes (GREST_Handle,
596 1                                           startMonthTime,
```

```
                                                endMonthTime,
                                                flags,
                                                1,
                                                times,
                                                &numberEntries,
                                                &cookie)) == E_SUCCESS)
  {

    /* If we got any back, then there are backups in that month */
    if (numberEntries > 0)
      backupsExist = BOOL_TRUE;
  }
  GUTIL_Free (times);

  return (backupsExist);
}
```



```
/*****************************************************************
 * REST_SetPreviousBackupInMonth
 *
 * Description:
 *   This routine will select the first backup (chronologically) done
 *   before the given day of the current month and year.
 *
 * Parameters:
 *   day (IO) - the day to start at, updated to found day (if any)
 *
 * Returns:
 *   BOOL_TRUE  - if a previous backup was found and selected
 *   BOOL_FALSE - otherwise
 *
 *****************************************************************/

static BoolEnum REST_SetPreviousBackupInMonth (Int *day)
{
  BoolEnum found = BOOL_FALSE;
                                        /* Flag if we found a previous backup */
  Int      tempDay;                     /* Temporary day holder */
  Int      index;                       /* Loop index */
  BoolEnum returnStatus = BOOL_FALSE;/* Status to return */

  /* Find the previous day with a backup */
  tempDay = *day - 1;
  while ((!found) && (tempDay >= 0))
  {
    if (backupTimesCount[tempDay] > 0)
    {
      /* This is the day */
      *day = tempDay;

      /* If there is more than one backup, use the latest backup */
      if (backupTimesCount[tempDay] > 1)
      {
        CBOX_Gold (backupTimeBox[tempDay], 0);
        CBOX_CurSelect (backupTimeBox[tempDay]);
      }

      /* Else use the only backup on this day */
      else
      {
        currentSelectedTime = backupTimes[tempDay][0];
      }
      found = BOOL_TRUE;
    }
    else
    {
      /* Go to the previous day */
      tempDay--;
    }
  }

  /* Return wheter or not we found it */
  return (found);
}
```

```
671
672   /*****************************************************************
673    * REST_CalendarSetPrevBackup
674    *
675    * Description:
676    *    This routine will select the previous backup (
677                                                  chronologically) to the
678    *    currently selected backup.
679    *
680    * Parameters:
681    *    None.
682    *
683    * Returns:
684    *    BOOL_TRUE  - If a previous backup was found and selected
685    *    BOOL_FALSE - otherwise
686    *
687    *****************************************************************/
688
689   BoolEnum REST_CalendarSetPrevBackup (void)
690   {
691     BoolEnum found = BOOL_FALSE;
692                                        /* Flag if we found a previous backup */
693     BoolEnum status;
694                                       /* Flag if previous exists for time */
695     Int      foundDay;                /* Day of previous backup */
696     Int      tempDay;                 /* Temporary day holder */
697     Int      tempMonth;               /* Temporary month holder */
698     Int      index;                   /* Loop index */
699     BoolEnum returnStatus = BOOL_FALSE;/* Status to return */
700     U_long   flags;
701
702     if (THUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
703       flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
704     else
705       flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;
706
707     /*
708      * Find the previous backup time
709      */
710
711     /* If we are in the selected month/year,
712                                          move from the current selection */
713     if ((displayedMonth == selectedMonth) &&
714         (displayedYear == selectedYear))
715     {
716
717       /* If this is the only backup or the earliest backup of the day */
718       index = (Int)CBOX_ChosenGetId (backupTimeBox[selectedDay]);
719       if ((backupTimesCount[selectedDay] == 1) ||
720           (index == backupTimesCount[selectedDay] - 1))
721       {
722
723         /* Find the previous day in the month with a backup */
724         foundDay = selectedDay;
725         found = REST_SetPreviousBackupInMonth (&foundDay);
726       }
727       else
728       {
729         /* There is an earlier backup on this day */
730         foundDay = selectedDay;
731         CBOX_GoId (backupTimeBox[selectedDay], index+1);
732         CBOX_CurSelect (backupTimeBox[selectedDay]);
733         found = BOOL_TRUE;
734       }
735     }
```



```
736     /* If we haven't found it yet, try the prev month */
737     if (!found)
738     {
739       /* First, check if a previous backup exists */
740       EDMRST_IsTherePrevBackupForTime (GREST_Handle,
741                                        currentSelectedTime,
742                                        flags,
743                                        &status);
744       if (status)
745       {
746
747         /* Find the previous month with a backup (we know one exists) */
748         tempMonth = displayedMonth - 1;
749         while (!found)
750         {
751
752           /* Only consider this month if backups were done in it */
753           if (REST_BackupExistsInMonth (tempMonth, displayedYear))
754           {
755             displayedMonth--;
756             REST_UpdateDisplayedDate();
757
758             /* get the last day of the month (
759                                                plus one so we can find the last) */
760             foundDay = GTIME_DaysPerMonth (
761                                                displayedMonth, displayedYear);
762             found = REST_SetPreviousBackupInMonth (&foundDay);
763           }
764           tempMonth--;
765
766         }
767       }
768     }
769
770     /* If we found one */
771     if (found)
772     {
773
774       /* Redraw the old selected day (has affect of unselecting) */
775       REST_DrawSelectedDay (BOOL_FALSE);
776
777       /* Set the selecte backup date */
778       selectedDay = foundDay;
779       selectedMonth = displayedMonth;
780       selectedYear = displayedYear;
781
782       /* Draw the newly selected day */
783       REST_DrawSelectedDay (BOOL_TRUE);
784
785       returnStatus = BOOL_TRUE;
786     }
787
788     return (returnStatus);
789   }
```

```
/*****************************************************************
 * REST_SetNextBackupInMonth
 *
 * Description:
 *   This routine will select the next backup (chronologically) done
 *   after the given day of the current month and year.
 *
 * Parameters:
 *   day (IO) - the day to start at, updated to found day (if any)
 *
 * Returns:
 *   BOOL_TRUE  - If a next backup was found and selected
 *   BOOL_FALSE - otherwise
 *
 *****************************************************************/

static BoolEnum REST_SetNextBackupInMonth (Int *day)
{
  BoolEnum found = BOOL_FALSE;
                                        /* Flag if we found a next backup */
  Int       tempDay;                    /* Temporary day holder */
  Int       index;                      /* Loop index */
  Int       numberDays;                 /* Number of days in the month */
  BoolEnum returnStatus = BOOL_FALSE;/* Status to return */

  /* Find the next day with a backup */
  tempDay = *day + 1;
  numberDays = GTIME_DaysPerMonth (displayedMonth, displayedYear);
  while ((!found) && (tempDay < numberDays))
  {
    if (backupTimesCount[tempDay] > 0)
    {
      /* This is the day */
      *day = tempDay;

      /* If there is more than one backup, use the first backup */
      if (backupTimesCount[tempDay] > 1)
      {
        CBOX_Gold (
                backupTimeBox[tempDay], backupTimesCount[tempDay] - 1);
        CBOX_CurSelect (backupTimeBox[tempDay]);
      }

      /* Else use the only backup on this day */
      else
      {
        currentSelectedTime = backupTimes[tempDay][0];
      }
      found = BOOL_TRUE;
    }
    else
    {
      /* Go to the next day */
      tempDay++;
    }
  }

  /* Return wheter or not we found it */
  return (found);
}
```



```
/*****************************************************************
 * REST_CalendarSetNextBackup
 *
 * Description:
 *   This routine will select the next backup (chronologically) to the
 *   currently selected backup.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   BOOL_TRUE  - If a next backup was found and selected
 *   BOOL_FALSE - otherwise
 *
 *****************************************************************/

BoolEnum REST_CalendarSetNextBackup (void)
{
  BoolEnum found = BOOL_FALSE;
                                        /* Flag if we found a previous backup */
  BoolEnum status;
                                                /* Flag if next exists for time */
  Int       foundDay;                   /* Day of previous backup */
  Int       tempDay;                    /* Temporary day holder */
  Int       tempMonth;                  /* Temporary month holder */
  Int       index;                      /* Loop index */
  Int       numberDays;                 /* Number of days in the month */
  BoolEnum returnStatus = BOOL_FALSE;/* Status to return */
  u_long    flags;

  if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
    flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
  else
    flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

  /*
   * Find the next backup time
   */

  /* If we are in the selected month/year,
                                        move from the current selection */
  if ((displayedMonth == selectedMonth) &&
      (displayedYear == selectedYear))
  {
    /* If this is the only backup or the last backup of the day */
    index = (Int)CBOX_ChosenGetId (backupTimeBox[selectedDay]);
    if ((backupTimesCount[selectedDay] == 1) ||
        (CBOX_ChosenGetId (backupTimeBox[selectedDay]) == 0))
    {

      /* Find the next day in the month with a backup */
      foundDay = selectedDay;
      found = REST_SetNextBackupInMonth (&foundDay);
    }
    else
    {
      /* There is a later backup on this day */
      index = (Int) CBOX_ChosenGetId (backupTimeBox[selectedDay]);
      foundDay = selectedDay;
      CBOX_Gold (backupTimeBox[selectedDay], index-1);
      CBOX_CurSelect (backupTimeBox[selectedDay]);
      found = BOOL_TRUE;
    }
  }
```

```
906 1   /* If we haven't found it yet, try the next month */
907 1   if (!found)
908 2   {


910 3     /* First, check if a previous backup exists */
911 2     EDMRST_IsThereNextBackupForTime (GREST_Handle,
912 2                                      currentSelectedTime,
913 2                                      flags,
914 2                                      &status);
915 2     if (status)
916 3     {

918 3       /* Find the previous next with a backup (we know one exists) */
919 3       tempMonth = displayedMonth + 1;
920 5       while (!found)
921 4       {

923 4         /* Only consider this month if backups were done in it */
924 4         if (REST_BackupExistsInMonth (tempMonth, displayedYear))
925 5         {
926 5           displayedMonth++;
927 5           REST_UpdateDisplayedDate();

929 5           /* use -1, day 0 should be checked */
930 5           foundDay = -1;
931 5           found = REST_SetNextBackupInMonth (&foundDay);
932 4         }
933 4         tempMonth++;
934 3       }
935 2     }
936 1   }

938 1   /* If we found one */
939 1   if (found)
940 2   {

942 2     /* Redraw the old selected day (has affect of unselecting) */
943 2     REST_DrawSelectedDay (BOOL_FALSE);

945 2     /* Set the selecte backup date */
946 2     selectedDay = foundDay;
947 2     selectedMonth = displayedMonth;
948 2     selectedYear = displayedYear;

950 2     /* Draw the newly selected day */
951 2     REST_DrawSelectedDay (BOOL_TRUE);

953 2     returnStatus = BOOL_TRUE;
954 .   }

956 1   return (returnStatus);
957 }
```



```
959 /*****************************************************************
960  * REST_CalendarDraw
961  *
962  * Description:
963  *   This routine will draw the calendar.
964  *
965  * Parameters:
966  *   None.
967  *
968  * Returns:
969  *   None.
970  *
971  *****************************************************************/

973 void REST_CalendarDraw (void)
974 1 {
975 1   Int          x;                          /* x location for drawing */
976 1   Int          y;                          /* y location for drawing */
977 1   Point16Rec   startPoint;                 /* Start point for line */
978 1   Point16Rec   endPoint;                   /* End point for line */
979 1   int          dayNumber;                  /* Day number to draw */
980 1   Rect16Rec    textRect;                   /* Bounding rectangle for text */
981 2   Point16Rec   textExt;                    /* Extents needed for text */
982 1   Char         dayString[SMALL_STRING_LENGTH]; /* String to draw */
983 1   Int          dayWidth;                   /* The width of a day box */
984 1   Int          dayHeight;                  /* The height of a day box */
985 1   int          numberDays;                 /* Number of days in the displayed month */
986 1   int          numberWeeks;                /* Number of weeks in the displayed month */

988 1   /* Get the width and height of the day */
989 1   REST_GetDayWidthHeight (&dayWidth, &dayHeight);

991 1   DRAW_SetColors ((WgtPtr)REST_CalendarWindow->CalendarPanel,
992 1                   COLOR_Black(),
993 1                   COLOR_Transparent());
994 1   DRAW_SetPen((WgtPtr)REST_CalendarWindow->CalendarPanel, PEN_Solid());

996 3   /* Draw the top line over the day labels */
997 1   startPoint.x = MARGIN_WIDTH;
998 1   startPoint.y = 0;
999 1   endPoint.x = startPoint.x + (dayWidth * DAYS_PER_WEEK);
1000 1  endPoint.y = startPoint.y;
1001 1  DRAW_Line ((
       WgtPtr)REST_CalendarWindow->CalendarPanel, &startPoint, &endPoint);

1003 1  /* Draw the day labels */
1004 1  DRAW_SetFont ((
                    WgtPtr)REST_CalendarWindow->CalendarPanel, FONT_Normal());
1005 1  for (x = 0; x<DAYS_PER_WEEK; x++)
1006 2  {
1007 2    STR_Sprintf (dayString, "%s", CTIME_GetDayStr(x+1));
1008 2    textRect.Ori.x = MARGIN_WIDTH + (x * dayWidth) + DAY_LABEL_OFFSET;
1009 2    textRect.Ori.y = 0;
1010 2    textRect.Ext.x = dayWidth;
```

```
1011 2      textRect.Ext.y = DAY_LABEL_HEIGHT;
1012 2      DRAW_Text ((WgtPtr)REST_CalendarWindow->CalendarPanel,
1013 2                 &textRect,
1014 2                 DRAW_JUSTCENTER,
1015 2                 dayString);
1016 1    }

1018 1    /* Determine the number of days and weeks in the displayed month */
1019 1    numberDays = GTIME_DaysPerMonth (displayedMonth, displayedYear);
1020 1    numberWeeks = GTIME_NumberWeeks (
                              displayedMonthTime.tm_wday, numberDays);

1022 1    /* Draw the vertical lines */
1023 1    for (x = 0; x<=DAYS_PER_WEEK; x++)
1024 2    {
1025 2      startPoint.x = MARGIN_WIDTH + (x * dayWidth);
1026 2      startPoint.y = 0;

1028 2      endPoint.x = startPoint.x;
1029 2      endPoint.y = DAY_LABEL_HEIGHT + (numberWeeks * dayHeight);
1030 2      DRAW_Line ((
          WgtPtr)REST_CalendarWindow->CalendarPanel, &startPoint, &endPoint);
1031 1    }

1033 1    /* Draw the horizontal lines */
1034 1    for (y = 0; y<=numberWeeks; y++)
1035 2    {
1036 2      startPoint.x = MARGIN_WIDTH;
1037 2      startPoint.y = DAY_LABEL_HEIGHT + (y * dayHeight);

1039 2      endPoint.x = MARGIN_WIDTH + (dayWidth * DAYS_PER_WEEK);
1040 2      endPoint.y = startPoint.y;
1041 2      DRAW_Line ((
          WgtPtr)REST_CalendarWindow->CalendarPanel, &startPoint, &endPoint);
1042 1    }

1044 1    /* Draw the date labels */
1045 1    dayNumber = displayedMonthTime.tm_wday;
1046 1    textRect.Ori.x = MARGIN_WIDTH + (
                                  dayNumber * dayWidth) + DAY_LABEL_OFFSET;
1047 1    textRect.Ori.y = DAY_LABEL_HEIGHT + DAY_LABEL_OFFSET;
1048 1    for (x = 1; x <= numberDays; x++)
1049 2    {
1050 2      STR_Sprintf (dayString, "%d", x);
1051 2      DRAW_SetFont ((
                WgtPtr)REST_CalendarWindow->CalendarPanel, FONT_Normal());
1052 3      DRAW_QueryTextExt((
            WgtPtr)REST_CalendarWindow->CalendarPanel, dayString, &textExt),
1053 2      textRect.Ext.x = textExt.x;
1054 2      textRect.Ext.y = textExt.y;

1056 2      DRAW_SetColors ((WgtPtr)REST_CalendarWindow->CalendarPanel,
1057 2                      COLOR_Black(),
1058 2                      COLOR_Transparent());
1059 2      DRAW_SetPen ((WgtPtr)REST_CalendarWindow->CalendarPanel, PEN_Solid(
                                                                    ));

1060 2      DRAW_Text ((WgtPtr)REST_CalendarWindow->CalendarPanel,
1061 2                 &textRect,
1062 2                 DRAW_JUSTCENTER,
1063 2                 dayString);

1065 2      /* Set up the next day */
1066 2      dayNumber++;

1068 2      /* If we have hit the end of the week */
1069 2      if (dayNumber == DAYS_PER_WEEK)
```



```
1070 3      {
1071 3        /* Set the x coords back to the start */
1072 3        dayNumber = 0;
1073 3        textRect.Ori.x = MARGIN_WIDTH + DAY_LABEL_OFFSET;

1075 3        /* Bump the y coords the height of a day */
1076 3        textRect.Ori.y += dayHeight;
1077 2      }
1078 2      else
1079 3      {
1080 3        /* Just bump the x coords the width of a day */
1081 3        textRect.Ori.x += dayWidth;
1082 2      }
1083 1    }

1085 1    /* Draw the selected Day */
1086 1    REST_DrawSelectedDay (BOOL_TRUE);

1088    }
```

```
1090  /*****************************************************************
1091   * REST_CalendarResize
1092   *
1093   * Description:
1094   *   This routine handles the resizing of the calendar.  It will then
1095   *   resize and relocate all combo boxes.
1096   *
1097   * Parameters:
1098   *   None.
1099   *
1100  * Returns:
1101  *   None.
1102  *
1103  *****************************************************************/

1105  void REST_CalendarResize (void)
1106  {
1107    Int           index;        /* loop counter */
1108    int           dayNumber;
                                    /* week day number for determining x position */
1109    Rect16Rec     textRect;     /* Rectangle to draw the text in */
1110    Point16Rec    textExt;      /* Extents necessary for the text */
1111    Char          dayString[SMALL_STRING_LENGTH];
                                    /* temporary string for test */
1112    Int           dayWidth;     /* The width of a day box */
1113    Int           dayHeight;    /* The height of a day box */
1114    Int           numberDays;
                                    /* Number of days in the displayed month */

1116    /* If we haven't initialized the boxes yet, get out */
1117    if (boxesInited != BOOL_TRUE)
1118    {
1119      return;
1120    }

1122    /* Get the width and height of the day */
1123    REST_GetDayWidthHeight (&dayWidth, &dayHeight);

1125    /* Determine the position for each of the choice boxes */
1126    /* Get a sample string to determin string height */
1127    DRAW_SetFont ((
                WgtPtr)REST_CalendarWindow->CalendarPanel, FONT_Normal());
1128    STR_Cpy (dayString, "88");
1129    DRAW_QueryTextRxt((
          WgtPtr)REST_CalendarWindow->CalendarPanel, dayString, &textExt);

1131    dayNumber = displayedMonthTime.tm_wday;
1132    textRect.Ori.x = MARGIN_WIDTH + dayNumber * dayWidth;
1133    textRect.Ori.y = DAY_LABEL_HEIGHT + DAY_LABEL_OFFSET + textExt.y;
1134    textRect.Ext.x = dayWidth;
1135    textRect.Ext.y = dayHeight - textExt.y - DAY_LABEL_OFFSET;

1137    /* Determine the number of days in the displayed month */
1138    numberDays = GTIME_DaysPerMonth (displayedMonth, displayedYear);

1140    /* Loop through all days and set up the box positions for that day
                                                                    */
1141    for (index = 0; index < numberDays; index++)
1142    {

1144      /* Position the time CBox and Text widget in the middle of the day
                                                                    */
1145      REST_CalSetBoxPos (index, textRect.Ori, textRect.Ext);

1147      /* Set up the next day */
```



```
1148      dayNumber++;

1150      /* If we have hit the end of the week */
1151      if (dayNumber == DAYS_PER_WEEK)
1152      {
1153        /* Set the x coords back to the start */
1154        dayNumber = 0;
1155        textRect.Ori.x = MARGIN_WIDTH;

1157        /* Bump the y coords the height of a day */
1158        textRect.Ori.y += dayHeight;
1159      }
1160      else
1161      {
1162        /* Just bump the x coords the width of a day */
1163        textRect.Ori.x += dayWidth;
1164      }
1165    }

1167  }
```

```
1169  /*****************************************************************
1170   * REST_CalSetBoxSelected
1171   *
1172   * Description:
1173   *   This routine will select the day and time for the given combo
                                                                  boxes
1174   *   currently selected time.
1175   *
1176   * Parameters:
1177   *   cbox (I) - The combo box which was selected
1178   *
1179   * Returns:
1180   *   None.
1181   *
1182   *****************************************************************/

1184  static void REST_CalSetBoxSelected (CBoxPtr cbox)
1185  {
1186    Int  newId; /* The new selection Id */

1188    /* Redraw the old selected day, unselected */
1189    REST_DrawSelectedDay (BOOL_FALSE);

1191    /* Get the new selected time */
1192    selectedDay = (Int) WGT_GetClientRes ((WgtPtr)cbox);
1193    selectedMonth = displayedMonth;
1194    selectedYear = displayedYear;
1195    newId = (Int) CBOX_ChosenGetId (cbox);
1196    currentSelectedTime = backupTimes[selectedDay][newId];

1198    /* Draw the newly selected day */
1199    REST_DrawSelectedDay (BOOL_TRUE);

1201  }
```



```
1203  /*****************************************************************
1204   * REST_SetDisplayedDate
1205   *
1206   * Description:
1207   *   This routine will display the current month and year in the date
1208   *   text area.
1209   *
1210   * Parameters:
1211   *   None.
1212   *
1213   * Returns:
1214   *   None.
1215   *
1216   *****************************************************************/

1218  static void REST_SetDisplayedDate (void)
1219  {
1220    Char  displayString[MEDIUM_STRING_LENGTH];  /* String to display */

1223    /* Set the month display string */
1223    STR_Sprintf (displayString, "%s", GTIME_GetMonthStr (
                                                  displayedMonth));
1224    TAREA_SetLabel (REST_CalendarWindow->MonthTArea, displayString);

1226    /* Set the year display string */
1227    STR_Sprintf (displayString, "%d", 1900 + displayedYear);
1228    TAREA_SetLabel (REST_CalendarWindow->YearTArea, displayString);
1229  }
```

```
/*****************************************************************
 * REST_UpdateDisplayedDate
 *
 * Description:
 *   This routine will update the displayed date to the current
                                                        displayed
 *   month and year.  It will redraw the calendar showing the backups
 *   for each day of the current month.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

static void REST_UpdateDisplayedDate (void)
{
  int         startDay = 1;
                                  /* Start day to get start month time */
  time_t      endMonthTime;       /* Last time for the month */
  time_t      startMonthTime;     /* First time for the month */
  struct tm *nextTime;
                                  /* Next time to examine in all times */
  struct tm  thisTime;            /* Temporary time storage */
  long       cookie = INIT_COOKIE;/* Ah, the magic cookie */
  short      numberEntries;       /* Number of times returned */
  time_t    *times;               /* Times returned from API */
  eerrno_ty  eerrno;              /* Returned error code */
  int        monthDay;            /* Day of month index from time */
  int        i;                   /* Loop counter */
  Char       timeString[MEDIUM_STRING_LENGTH];  /* Converted string */
  BoolEnum   first;
                                  /* Flag if this is the first time for day */
  u_long     flags;

  if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
    flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
  else
    flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

  /* Update the current date */
  GTIME_UpdateDate (&startDay,
                    &displayedMonth,
                    &displayedYear,
                    BOOL_FALSE);
  REST_SetDisplayedDate ();

  /* Get the start time for the month */
  GTIME_GetTimeTm (startDay,
                   displayedMonth,
                   displayedYear,
                   0,
                   0,
                   &displayedMonthTime);
  startMonthTime = mktime (&displayedMonthTime);

  /* Get the end time for the month */
  GTIME_GotTimeTm (GTIME_DaysPerMonth(displayedMonth, displayedYear),
                   displayedMonth,
                   displayedYear,
                   HOURS_PER_DAY - 1,
                   MINUTES_PER_HOUR - 1,
                   &thisTime);
```



```
  endMonthTime = mktime (&thisTime);

  /* Re-initialize the counts, and the cboxes */
  for (i=0; i<MAX_DAYS_PER_MONTH; i++)
  {
    backupTimesCount[i] = 0;
    CBOX_GoFirst(backupTimeBox[i]);
    while (CBOX_IsOk (backupTimeBox[i]))
      CBOX_CurRemoveElt (backupTimeBox[i]);
    WGT_SetInvisible ((WgtPtr)backupTimeBox[i]);
    WGT_SetInvisible ((WgtPtr)backupTimeText[i];
  }

  /* Flag that we are selecting CBox values by code to stop CB
                                                        processing */
  selectionByCode = BOOL_TRUE;

  /* Allocate space for the times array */
  times = (time_t *) GUTIL_Malloc (TIME_BUFFER_LENGTH * sizeof(
                                                        time_t));


  /* Until the API says we're done, keep looping getting all times */
  while (cookie != DONE_COOKIE)
  {
    /* Get the next batch of times */
    if ((eerrno = EDMRST_GetAllBackupTimes (GREST_Handle,
                                            startMonthTime,
                                            endMonthTime,
                                            flags,
                                            TIME_BUFFER_LENGTH,
                                            times,
                                            &numberEntries,
                                            &cookie)) == E_SUCCESS)
    {

      /* Loop through all the backup times */
      for (i = 0; i < numberEntries; i++)
      {

        /* Get the time struct so we know the day of the month */
        nextTime = localtime (&times[i]);

        /* Get the day of the month index into the arrays */
        monthDay = nextTime->tm_mday - 1;

        /* Make sure we have space for this time */
        if (backupTimesCount[monthDay] < MAX_TIME_SLOTS)
        {

          /* Store this time in the right place in the array of times
                                                                  */
          backupTimes[monthDay][backupTimesCount[monthDay]] = times[i];

          /* Get the time in a displayable format */
          strftime (timeString,
                    MEDIUM_STRING_LENGTH,
                    "%i:%M",
                    localtime(&times[i]));

          /* Since we have a time for the date,
                                          make the choice box visible */
          if (backupTimesCount[monthDay] == 0)
          {
            WGT_SetVisible ((WgtPtr)backupTimeText[monthDay]);
            TED_SetStr (backupTimeText[monthDay], timeString);
          }
```

```
1354 5            else
1355 6            {
1356 6              WGT_SetInvisible ((WgtPtr)backupTimeText[monthDay]);
1357 6              WGT_SetVisible ((WgtPtr)backupTimeBox[monthDay]);
1358 5            }

1360 5            /*
1361 5             * NOTE:
1362 5             *
1363 5             *  The times are given to us in descending order.
                                                          We want to
1364 5             *  put them in the list such that the first is the earliest
                                                              and
1365 5             *  the last is the latest.
                                                 If we go to the ID of the last one
1366 5             *  we got and do a CBOX_CurAddElt it will put it before it
                                                                        so
1367 5             *  that will reverse the order for us.
                                                        The latest backup on
1368 5             *  that day will then be 0, and the earliest will then be
1369 5             *  backupTimesCount[date].
1370 5             */

1372 5            /* Get to the end of the list */
1373 5            if (backupTimesCount[monthDay] > 0)
1374 6            {
1375 6              first = BOOL_FALSE;
1376 6              CBOX_GoId (backupTimeBox[monthDay],
1377 6                         backupTimesCount[monthDay]);
1378 6            }
1379 5            else
1380 6            {
1381 6              first = BOOL_TRUE;
1382 6              CBOX_GoFirst(backupTimeBox[monthDay]);
1384 5            }

1385 5            /* Add this time to the element list */
1386 5            CBOX_CurAddElt (backupTimeBox[monthDay], NULL);

1388 5            /* Set the label for this element to the time string */
1389 5            CBOX_CurSetLabel (backupTimeBox[monthDay], timeString);

1391 5            /* Set the ID to the index into the backup times */
1392 5            CBOX_CurSetId (backupTimeBox[monthDay],
1393 5                           backupTimesCount[monthDay]);

1395 5            /* If this is the current time or the first for that day.
                                                              select it */
1396 5            if ((times[i] == currentSelectedTime) || first)
1397 5              CBOX_CurSelect (backupTimeBox[monthDay]);

1399 5            /* Also select the day */
1400 5            if (times[i] == currentSelectedTime)
1401 5              RBST_CalSetBoxSelected (backupTimeBox[monthDay]);

1403 5            /* Update the time count for this day */
1404 5            backupTimesCount[monthDay]++;
1405 4          }
1406 3        }

1408 2      }
1409 2      else
1410 3      {
1411 3        /* Nothing more to see here folks... Nothing more to see... */
1412 3        cookie = DONE_COOKIE;
1413 2      }
```



```
1414 1    }

1416 1    /* Free the times array, we be done with it */
1417 1    GUTIL_Free (times);

1419 1    /* Remove the selection by code flag */
1420 1    selectionByCode = BOOL_FALSE;

1422 1    /* Resize the window, for different number of weeks in this month */
1423 1    RBST_CalendarResize ();

1425 1    /* Invalidate the calendar to force a redraw */
1426 1    WGT_Inval ((WgtPtr)RBST_CalendarWindow->CalendarPanel, BOOL_TRUE);

1428    }
```

```
1430 /*****************************************************************
1431  * REST_CalendarGetMostRecentTime
1432  *
1433  * Description:
1434  *   This routine will determine the time of the most recent backup.
1435  *
1436  * Parameters:
1437  *   None.
1438  *
1439  * Returns:
1440  *   None.
1441  *
1442  *****************************************************************/

1444 time_t REST_CalendarGetMostRecentTime (void)
1445 {
1446   time_t      mostRecentTime = 0;  /* Time of most recent backup */
1447   int         i;                   /* Loop index */
1448   long        cookie = INIT_COOKIE;/* Ah, the magic cookie */
1449   short       numberEntries;       /* Number of times returned */
1450   time_t     *times;               /* Array of times returned */
1451   eerrno_ty   eerrno;              /* Error code returned */
1452   u_long      flags;

1454   if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
1455     flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
1456   else
1457     flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

1459   /* Allocate space for the times */
1460   times = (time_t *) GUTIL_Malloc (TIME_BUFFER_LENGTH * sizeof(
                                                        time_t));

1462   /* Keep looping until no more times are found */
1463   while (cookie != DONE_COOKIE)
1464   {

1466     /* Get all times (
                we can start with the current selection to minimize) */
1467     if ((eerrno = EDMRST_GetAllBackupTimes (GREST_Handle,
1468                                             currentSelectedTime,
1469                                             time(NULL),
1470                                             flags,
1471                                             TIME_BUFFER_LENGTH,
1472                                             times,
1473                                             &numberEntries,
1474                                             &cookie)) == 0)
1475     {

1477       /* Loop through all the backup times */
1478       for (i = 0; i < numberEntries; i++)
1479       {

1481         /* Check if this is more recent then the current */
1482         if (times[i] > mostRecentTime)
1483           mostRecentTime = times[i];
1484       }

1486     }
1487     else
1488     {
1489       /* Nothing more to see here folks... Nothing more to see... */
1490       cookie = DONE_COOKIE;
1491     }
1492   }
```



```
1494   /* Free the times array */
1495   GUTIL_Free (times);

1497   return (mostRecentTime);
1498 }
```

```
/*****************************************************************
 * REST_CalendarUpdateDate
 *
 * Description:
 *   This routine will handle the callback to update the displayed
 *   month based on the current selection in the Month and Year
 *   combo boxes.
 *
 * Parameters:
 *   newMonth      (I) - The new month number to display
 *   newYear       (I) - The new year number to display
 *   isMonthUpdate (I) - Flag if this update is a month update
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

static void REST_CalendarUpdateDate (int       newMonth,
                                     int       newYear,
                                     BoolEnum  isMonthUpdate)
{
  int       startDay = 1;
                        /* Temporary storage for first day of month */
  int       tempMonth;    /* Temporary month storage for new month */
  int       tempYear;     /* Temporary year storage */
  struct tm thisTime;
                        /* Time struct for last time in previous month */
  time_t    endMonthTime;  /* Last time in previous month */
  BoolEnum  previous;
                        /* Flag if the selected is prior to current */
  int       status = 0;   /* Return status from GREST calls */
  boolean_ty isThere = 0;  /* Flag if the backup exists */
  int       currentDay;   /* Today's day */
  int       currentMonth;  /* Today's month */
  int       currentYear;   /* Today's year */
  Char      outputString[MAX_STRING_LENGTH];
                        /* Error string to display */
  time_t    mostRecentTime = 0;  /* Time of most recent backup */
  int       tmpDay;       /* Temporary storage */
  int       tmpHour;      /* Temporary storage */
  int       tmpMinutes;   /* Temporary storage */
  u_long    flags;
  BoolEnum  dateFault = BOOL_FALSE;
                        /* Flag if the date could not be changed */

  if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
    flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
  else
    flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

  /* Get the adjusted month and year */
  tempMonth = newMonth;
  tempYear = newYear;
  GTIME_UpdateDate (&startDay,
                    &tempMonth,
                    &tempYear,
                    BOOL_FALSE);

  /*
   * Determine if the user is going past the current day, don't allow
   * it since there can't be any future backups (I hope)
   */

  /* Get the current date */
  GTIME_GetCurrentDate (&currentDay, &currentMonth, &currentYear);
  if ((tempYear > currentYear) ||
      ((tempYear == currentYear) && (tempMonth > currentMonth)))
  {
    /*
     * If this is not a month update, determine the most recent month
     * and go there
     */

    if (!isMonthUpdate)
    {
      /* Display the error message */
      GALSRT_DisplayError ((WinPtr)REST_CalendarWindow,
                           REST_GetErrorString (REST_WARNING_INDEX),
                           GICON_GetWarning(),
                           REST_GetErrorString (
                                              REST_NO_FUTURE_MESSAGE));

      mostRecentTime = REST_CalendarGetMostRecentTime ();

      /* If there is a more recent time */
      if (mostRecentTime != 0)
      {
        /* Get the components of the time */
        GTIME_BreakDownTime (mostRecentTime,
                             &tmpDay,
                             &displayedMonth,
                             &displayedYear,
                             &tmpHour,
                             &tmpMinutes);
        REST_UpdateDisplayedDate();
      }

      dateFault = BOOL_TRUE;
    }
  }

  /* Determine if this is a previous or next operation */
  if ((tempYear < displayedYear) ||
      ((tempYear == displayedYear) && (tempMonth < displayedMonth)))
    previous = BOOL_TRUE;
  else
    previous = BOOL_FALSE;

  /* Get the last time in the month */
  GTIME_GetTimeTm (GTIME_DaysPerMonth(tempMonth, tempYear),
                   tempMonth,
                   tempYear,
                   HOURS_PER_DAY - 1,
                   MINUTES_PER_HOUR - 1,
                   &thisTime);
  endMonthTime = mktime (&thisTime);

  /* If there are no backups in this or any previous/next month,
                                                        warn user */
  if (!dateFault && !REST_BackupExistsInMonth (tempMonth, tempYear))
  {
    if (previous)
    {
      status = EDMRST_IsTherePrevBackupForTime (GREST_Handle,
                                                endMonthTime,
                                                flags,
                                                &isThere);

      if ((status == E_SUCCESS) && !isThere)
      {
        STR_Sprintf (outputString,
                     REST_GetErrorString (REST_NO_PREVIOUS_FORMAT),
```

```
1624 4                          GTIME_GetMonthStr(tempMonth),
1625 4                          1900 + tempYear);

1627 4          /* Display the error message */
1628 4          GALERT_DisplayError ((WinPtr)REST_CalendarWindow,
1629 4                               REST_GetErrorString (REST_WARNING_INDEX),
1630 4                               GICON_GetWarning(),
1631 4                               outputString);

1633 4          dateFault = BOOL_TRUE;
1634 3        }
1635 3        else if (status != E_SUCCESS)
1636 4        {
1637 4  /*
1638 4          printf (
                    "Error returned from EDMRST_IsTherePrevBackupForTime\n");
1639 4  */
1640 3        }
1641 2      }
1642 2      else
1643 3      {
1644 3        status = EDMRST_IsThereNextBackupForTime (GREST_Handle,
1645 3                                                  endMonthTime,
1646 3                                                  flags,
1647 3                                                  &isThere);
1648 3        if ((status == E_SUCCESS) && !isThere)
1649 4        {
1650 4          STR_Sprintf (outputString,
1651 4                       REST_GetErrorString (REST_NO_NEXT_FORMAT),
1652 4                       GTIME_GetMonthStr(tempMonth),
1653 4                       1900 + tempYear);

1655 4          /* Display the error message */
1656 4          GALERT_DisplayError ((WinPtr)REST_CalendarWindow,
1657 4                               REST_GetErrorString (REST_WARNING_INDEX),
1658 4                               GICON_GetWarning(),
1659 4                               outputString);

1661 4          dateFault = BOOL_TRUE;
1662 3        }
1663 3        else if (status != E_SUCCESS)
1664 4        {
1665 4  /*
1666 4          printf (
                    "Error returned from EDMRST_IsThereNextBackupForTime\n");
1667 4  */
1668 3        }
1669 2      }
1670 1    }

1672 1    if (!dateFault)
1673 2    {
1674 2      /* Update the displayed month and year */
1675 2      displayedMonth = tempMonth;
1676 2      displayedYear = tempYear;
1677 2      REST_UpdateDisplayedDate();
1678 1    }
1679  }
```



```
1681  /*****************************************************************
1682   * REST_CalendarPreviousMonth
1683   *
1684   * Description:
1685   *   This routine will handle the callback to update the displayed
1686   *   month to the previous month.
1687   *
1688   * Parameters:
1689   *   None.
1690   *
1691   * Returns:
1692   *   None.
1693   *
1694   *****************************************************************/

1696  void REST_CalendarPreviousMonth (void)
1697 1 {
1698 1   REST_CalendarUpdateDate (
                                displayedMonth - 1, displayedYear, BOOL_TRUE);

1699  }
```

```
1701  /*****************************************************************
1702   * REST_CalendarNextMonth
1703   *
1704   * Description:
1705   *   This routine will handle the callback to update the displayed
1706   *   month to the next month.
1707   *
1708   * Parameters:
1709   *   None.
1710   *
1711   * Returns:
1712   *   None.
1713   *
1714   *****************************************************************/

1716  void REST_CalendarNextMonth (void)
1717  1 {
1718  1   REST_CalendarUpdateDate (
                                displayedMonth + 1, displayedYear, BOOL_TRUE);
1719    }
```



```
1721  /*****************************************************************
1722   * REST_CalendarPreviousYear
1723   *
1724   * Description:
1725   *   This routine will handle the callback to update the displayed
1726   *   Year to the previous Year.
1727   *
1728   * Parameters:
1729   *   None.
1730   *
1731   * Returns:
1732   *   None.
1733   *
1734   *****************************************************************/

1736  void REST_CalendarPreviousYear (void)
1737  1 {
1738  1   REST_CalendarUpdateDate (
                                displayedMonth, displayedYear - 1, BOOL_FALSE);
1739    }
```

```
1741  /*************************************************************************
1742   * REST_CalendarNextYear
1743   *
1744   * Description:
1745   *   This routine will handle the callback to update the displayed
1746   *   Year to the next Year.
1747   *
1748   * Parameters:
1749   *   None.
1750   *
1751   * Returns:
1752   *   None.
1753   *
1754   *************************************************************************/

1756  void REST_CalendarNextYear (void)
1757 1 {
1758 1   REST_CalendarUpdateDate (
                                displayedMonth, displayedYear + 1, BOOL_FALSE);
1759  }
```



```
1761  /*************************************************************************
1762   * REST_CalendarToday
1763   *
1764   * Description:
1765   *   This routine will handle the most recent button callback, it will
1766   *   display the month with the most recent backup and will select the
1767   *   most recent backup.
1768   *
1769   * Parameters:
1770   *   None.
1771   *
1772   * Returns:
1773   *   None.
1774   *
1775   *************************************************************************/

1777  void REST_CalendarToday (void)
1778 1 {
1779 1   time_t      mostRecentTime = 0;   /* Time of most recent backup */
1780 1   int         selectedHour;         /* Temporary storage */
1781 1   int         selectedMinutes;      /* Temporary storage */

1783 1   mostRecentTime = REST_CalendarGetMostRecentTime ();

1785 1   /* If there is a more recent time */
1786 1   if (mostRecentTime != 0)
1787 2   {

1789 2     /* Redraw the old selected day (has affect of unselecting) */
1790 2     REST_DrawSelectedDay (BOOL_FALSE);

1792 2     /* Set the selected time to the most recent backup */
1793 2     currentSelectedTime = mostRecentTime;

1795 2     /* Get the components of the time */
1796 2     GTIME_BreakDownTime (currentSelectedTime,
1797 2                          &selectedDay,
1798 2                          &selectedMonth,
1799 2                          &selectedYear,
1800 2                          &selectedHour,
1801 2                          &selectedMinutes);

1803 2     /* Display the most recent date */
1804 2     displayedMonth = selectedMonth;
1805 2     displayedYear = selectedYear;
1806 2     REST_UpdateDisplayedDate();


1809 2     /* Draw the newly selected day */
1810 2     REST_DrawSelectedDay (BOOL_TRUE);
1811 1   }

1813  }
```

```
1815  /*****************************************************************
1816   * REST_CalendarCancel
1817   *
1818   * Description:
1819   *    This routine will cancel the restore calendar window.  It will
1820   *    set the selected time to time 0, indicating the cancel action.
1821   *
1822   * Parameters:
1823   *    None.
1824   *
1825   * Returns:
1826   *    BOOL_TRUE  - If the cancel was effective
1827   *    BOOL_FALSE - If the cancel was already in place
1828   *
1829   *****************************************************************/
1831  BoolEnum REST_CalendarCancel (void)
1832  1 {
1834  1   /* If we haven't terminated yet, cancel */
1835  1   if (!REST_CalTerminated)
1836  2   {
1837  3     currentSelectedTime = 0;
1838  2     WIN_ModalReturn((WinPtr)REST_CalendarWindow, (ClientPtr)NULL);
1839  1   }
1842  1   return (!REST_CalTerminated);
1843    }
```



```
1845  /*****************************************************************
1846   * REST_CalendarOK
1847   *
1848   * Description:
1849   *    This routine will clout the restore calendar window with a new
1850   *    selected time.
1851   *
1852   * Parameters:
1853   *    None.
1854   *
1855   * Returns:
1856   *    None.
1857   *
1858   *****************************************************************/
1860  void REST_CalendarOK (void)
1861  1 {
1863  1   WIN_ModalReturn((WinPtr)REST_CalendarWindow, (ClientPtr)NULL);
1865    }
```

```
1867  /*****************************************************************
1868   * REST_DateBoxNfy
1869   *
1870   * Description:
1871   *   This routine handles all notifications for the combo boxes.
1872   *
1873   * Parameters:
1874   *   cbox (I) - The combo box receiving the notification.
1875   *   code (I) - The notification code received.
1876   *
1877   * Returns:
1878   *   None.
1879   *
1880   *****************************************************************/

1881  static  void  REST_DateBoxNfy (CBoxPtr      cbox,
1883                                 CBoxNfyEnum code)
1884 1 {
1885 1   Rect16Rec  box;  /* Widget box for the cbox */

1887 2   switch (code) {
1888 2   case CBOX_NFYELTSELECTED:
1889 2     CBOX_DefNfy(cbox, code);
1890 2     /* USER CODE */
1891 2     if (!selectionByCode)
1892 2       REST_CalSetBoxSelected (cbox);
1893 2     break;
1894 2   case CBOX_NFYGAINFOCUS:
1895 2   case CBOX_NFYFOCUSLOST:
1896 2     break;
1897 2   case CBOX_NFYINIT:
1898 2     CBOX_DefNfy(cbox, code);
1899 2     box.Ori.x = 0;
1900 2     box.Ori.y = 0;
1901 2     box.Ext.x = DATE_CBOX_WIDTH;
1902 2     box.Ext.y = DATE_CBOX_HEIGHT;
1903 2     WGT_Configure ((WgtPtr)cbox, &box.Ori, &box.Ext);
1904 2     break;
1905 2   default:
1906 2     CBOX_DefNfy(cbox, code);
1907 1   }

1909  }
```



```
1911  /*****************************************************************
1912   * REST_GetUserSelectedTime
1913   *
1914   * Description:
1915   *   This routine display a calendar which allows the user to select
1916   *   a backup time.  If the user selects a time and hits OK, the time
1917   *   selected will be returned,
1918   *                       if the user cancels time 0 will be returned.
1919   *
1920   * Parameters:
1921   *   currentWI   (I) - The current work-item to get the time for.
1922   *   currentTime (I) - The current time for the work-item
1923   *   parentWin   (
1924   *                I) - The parent window to make the calendar a child of.
1925   *
1926   * Returns:
1927   *   time_t representing the selected time
1928   *   0 if user cancelled or if an error occurs
1929   *
1930   *****************************************************************/

1931  time_t  REST_GetUserSelectedTime (GREST_Object currentWI,
1932                                    time_t       currentTime,
1933                                    WinPtr       parentWin)

1934 1 {
1935 1   int   selectedHour;                    /* Temporary storage */
1936 1   int   selectedMinutes;                 /* Temporary storage */
1937 1   Char  name[GMAX_OBJECT_LENGTH];        /* Name of the work item */
1938 1   Int   index;                           /* Loop Counter */
1939 1   Int   month;
                                   /* Month counter to fill cbox with */
1940 1   Int   year;
                                    /* Year counter to fill cbox with */
1941 1   Char  displayString[GMAX_OBJECT_LENGTH]; /* Displayed year string */
1942 1   int   currentDay;                       /* Today's day */
1943 1   int   currentMonth;                     /* Today's month */
1944 1   int   currentYear;                      /* Today's year */

1946 1   /* Flag that the boxes haven't been initialized yet */
1947 1   boxesInited = BOOL_FALSE;

1949 1   /* Load and initialize the window resources */
1950 1   REST_CalendarLoadInit (parentWin);

1952 1   /* Create the widgets for each day */
1953 1   for (index = 0; index < MAX_DAYS_PER_MONTH; index++)
1954 2   {
1955 2     /* Create a dropdown combo box for the day (
                                          to show multiple times) */
1956 2     backupTimeBox[index] = REST_CalCreateDateBox ();
1957 2     PANEL_AddWgt (REST_CalendarWindow->CalendarPanel, (
                                    WgtPtr)backupTimeBox[index]);
1958 2     WGT_SetNfyProc ((WgtPtr)backupTimeBox[index], REST_DateBoxNfy);
1959 2     WGT_SetClientRes ((WgtPtr)backupTimeBox[index], (ClientPtr)index);
1960 2     WGT_SetInvisible ((WgtPtr)backupTimeBox[index]);

1962 2     /* Create a TEd for the day (to show single times) */
1963 2     backupTimeText[index] = REST_CalCreateDateText ();
1964 2     PANEL_AddWgt (REST_CalendarWindow->CalendarPanel, (
                                    WgtPtr)backupTimeText[index]);
1965 2     WGT_SetClientRes ((WgtPtr)backupTimeText[index], (ClientPtr)index);
1966 2     WGT_SetInvisible ((WgtPtr)backupTimeText[index]);
1967 1   }
```

```
1969 1    /* Get the current date */
1970 1    GTIME_GetCurrentDate (&currentDay, &currentMonth, &currentYear);
1971 1
1972 1    /* Flag that we have initialized the boxes */
1973 1    boxesInited = BOOL_TRUE;
1974 1
1975 1    /* If the called passed a time, use it */
1976 1    if (currentTime != 0)
1977 1      currentSelectedTime = currentTime;
1978 1
1979 1    /* Otherwise, use the current backup time */
1980 1    else if (EDMRST_GetCurrentBackupTime (GREST_Handle,
1981 1                                          &currentSelectedTime) !=
                                                                E_SUCCESS)
1982 2    {
1983 2      return (0);
1984 1    }
1985 1
1986 1    /* Break down the time to get the current month and year to display
                                                                        */
1987 1    GTIME_BreakDownTime (currentSelectedTime,
1988 1                         &selectedDay,
1989 1                         &selectedMonth,
1990 1                         &selectedYear,
1991 1                         &selectedHour,
1992 1                         &selectedMinutes);
1993 1    displayedMonth = selectedMonth;
1994 1    displayedYear  = selectedYear;
1995 1
1996 1    /* Set the window title to the default settings */
1997 1    STR_Cpy (name, EDMRST_GetObjectFullName (GREST_Handle, currentWI));
1998 1    GUTIL_SetDefaultWindowTitle ((WinPtr)REST_CalendarWindow, name);
1999 1
2000 1    /* Display the current date */
2001 1    REST_UpdateDisplayedDate ();
2002 1
2003 1    /* Reset the terminated flag */
2004 1    REST_CalTerminated = BOOL_FALSE;
2005 1
2006 1    /* Position the window and go on our merry way */
2007 1    GWIN_CenterWindowInParent ((WinPtr)REST_CalendarWindow);
2008 1    WIN_ModalProcess ((WinPtr)REST_CalendarWindow);
2009 1
2010 1    /* Flush the events */
2011 1    EVENT_ProcessPending ();
2012 1
2013 1    /* Flag that we are done, and terminate the window */
2014 1    REST_CalTerminated = BOOL_TRUE;
2015 1    WIN_Terminate ((WinPtr)REST_CalendarWindow);
2016 1
2017 1    /* return the selected time */
2018 1    return (currentSelectedTime);
2019    }
```

```
/* -- Template created by NEURON DATA Open Interface.
/* -- Do not alter 'CodeGen' directives.                              */
/* {{ CodeGen: GeneratorVersion 4 }}                                  */
/* -- Code generated on 09/03/99 at 10:48:08.
/* {{ CodeGen: CodeHistory }}                                         */
#define ERR_LIB RESTORE
#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <util/esl_core.h>
#include <eerrno/e_errno.h>
#include <util/esl_string.h>

/* WARNING: UNIX DEPENDENCY!!! Used for polling sockets */
#include <stropts.h>
#include <poll.h>
#include <unistd.h>
#include "fbrowser/epcomm_api.h"
#include "util/hyper.h"

#include <restore/restore_api.h>

#include "restDest.h"
#include "restoreP.h"
#include "restUtils.h"
#include "restSelMgr.h"
#include "gutil/winUtils.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "gutil/iconDefs.h"
#include "gutil/iconUtils.h"
#include "gutil/alertMgr.h"


ERR_EXTERN
ERR_INMODULE("restDest")

/* {{ CodeGen: ClassImplementationPlaceHolder }}                      */
/* {{ CodeGen: WinClassImplementationPlaceHolder }}                   */

/*****************************************************************
 * REST_VerifySpace
 *
 * Description:
 *    This routine will verify the amount of space for the restore.
 *
 * Parameters:
 *    win        (I) - Parent window
 *    hostName   (I) - Name of the destination host
 *    directory (I) - The destination directory path
 *
 * Returns:
 *    BOOL_TRUE  - If it is OK to proceed with restore
 *    BOOL_FALSE - If there is not enough space and the user cancelled
 *
 *****************************************************************/
```



```
static BoolEnum REST_VerifySpace (RestDestWinPtr win,
                                  Str            hostName,
                                  Str            directory)
{
  struct fs_entry *entries;
                                        /* Mount point entries received */
  int             status;               /* Return status */
  Char            restoreString[64];
                                        /* String for space needed */
  Char            availableString[64];
                                        /* String for space available */
  u_hyper         restoreSize;
                                        /* Space needed for restoral */
  u_hyper         kSize;
                                        /* 1K representation as hyper */
  u_hyper         availableSize;        /* Space available */
  long            restoreFiles;
                                     /* Number of files to be restored */
  long            badFiles;          /* Number of bad files */
  BoolEnum        OKtoRestore = BOOL_TRUE;
                                        /* Flag if OK to proceed */


  /* Validate the parameters */
  if ((hostName != NULL) && (STR_Len (hostName) > 0) &&
      (directory != NULL) && (STR_Len (directory) > 0))
  {

    /* Get the filesystem info for the destination */
    if ((rpc_get_fs_info (hostName,
                          directory,
                          &entries,
                          &status) == SVC_CALL_SUCCEEDED) &&
        (status == E_SUCCESS) &&
        (entries != NULL))
    {
      /* Get the restore size in K */
      EDMRST_GetMarkedTotalSize (GREST_Handle, &restoreSize);
      kSize = ul_to_uh ((unsigned long)1024);
      u_hyper_divide_by (&restoreSize, kSize);

      /* Get the space available (as a hyper) */
      if (entries->kbytes_avail > 0)
        availableSize = ul_to_uh ((
                                   unsigned long) entries->kbytes_avail);
      else
        availableSize = ul_to_uh ((unsigned long) 0);

      /* IF the restore size is greater than the available space */
      if (u_hyper_greater_than (restoreSize, availableSize))
      {
        Char  outputString[MAX_STRING_LENGTH];
                                        /* String to display */

        /* Create the error question */
        REST_SPrintHyper (restoreString, restoreSize);
        REST_SPrintHyper (availableString, availableSize);
        STR_Sprintf (outputString,
                     REST_GetErrorString (REST_SPACE_ERROR_FORMAT),
                     restoreString,
                     availableString);

        /* Ask the user if he/she wants to continue anyways */
        if (GALERT_DisplayQuestion((WinPtr)win,
                                   REST_GetErrorString (
                                                REST_SPACE_ERROR_TITLE),
```

```
                                        GICON_GetWarning(),
                                        outputString,
                                        BOOL_FALSE) != GALERT_Affirmative)

        {
          /* The user cancelled the restore */
          OKtoRestore = BOOL_FALSE;
        }
      }

    /* IF it is still OK to restore */
    if (OKtoRestore)
    {

      /* Get the number of marked files */
      REST_GetCurrentMarkedInfo (&restoreFiles, &badFiles);

      /* IF there is more files then available files (inodes) */
      if (restoreFiles > entries->files_free)
      {
        Char  outputString[MAX_STRING_LENGTH];
                                              /* String to display */

        /* Create the error question */
        STR_Sprintf (restoreString, "%ld", restoreFiles);
        STR_Sprintf (availableString, "%ld", entries->files_free);
        STR_Sprintf (outputString,
                     REST_GetErrorString (REST_INODE_ERROR_FORMAT),
                     restoreString,
                     availableString);

        /* Ask the user if he/she wants to continue anyways */
        if (GALERT_DisplayQuestion((WinPtr)win,
                                   REST_GetErrorString(
                                             REST_INODE_ERROR_TITLE),
                                    GICON_GetWarning(),
                                    outputString,
                                    BOOL_FALSE) !=
                                              GALERT_Affirmative)

        {
          /* The user cancelled the restore */
          OKtoRestore = BOOL_FALSE;
        }
      }
    }

    /* Free the entries returned */
    free_fs_info (&entries);
  }

  /* Return whether or not it is OK to proceed with the restore */
  return (OKtoRestore);
}

/* {{ CodeGen: WindowSection Win
                                                                      */
/* ======================================================================
                                                                      */
/* == Code for Window "Win"
                                                                      */
/* ======================================================================
                                                                      */
/* {{ CodeGen: MenuImplementationPlaceHolder }}
```



```
                                                                      */

/* {{ CodeGen: WgtNfyHandler HitAlwaysButtone
                                                                      */
static   void C_FAR RestDestWin_HitAlwaysButtone L1(
                                                    RestDestWinPtr, win)
{
}
/* }} CodeGen: WgtNfyHandler HitAlwaysButtone
                                                                      */

/* {{ CodeGen: WgtNfyHandler HitOlderButton
                                                                      */
static   void C_FAR RestDestWin_HitOlderButton L1(RestDestWinPtr, win)
{
}
/* }} CodeGen: WgtNfyHandler HitOlderButton
                                                                      */

/* {{ CodeGen: WgtNfyHandler HitNeverButton
                                                                      */
static   void C_FAR RestDestWin_HitNeverButton L1(RestDestWinPtr, win)
{
}
/* }} CodeGen: WgtNfyHandler HitNeverButton
                                                                      */

/* {{ CodeGen: WgtNfyHandler HitNetworkButton
                                                                      */
static   void C_FAR RestDestWin_HitNetworkButton L1(
                                                    RestDestWinPtr, win)
{
  REST_UpdateDataPath (win);
}
/* }} CodeGen: WgtNfyHandler HitNetworkButton
                                                                      */

/* {{ CodeGen: WgtNfyHandler HitSymmConnButton
                                                                      */
static   void C_FAR RestDestWin_HitSymmConnButton L1(
                                                    RestDestWinPtr, win)
{
  REST_UpdateDataPath (win);
}
/* }} CodeGen: WgtNfyHandler HitSymmConnButton
                                                                      */

/* {{ CodeGen: WgtNfyHandler HitInPlaceButton
                                                                      */
static   void C_FAR RestDestWin_HitInPlaceButton L1(
                                                    RestDestWinPtr, win)
{
  REST_UpdateDestinationSensitivity (win);
}
/* }} CodeGen: WgtNfyHandler HitInPlaceButton
                                                                      */

/* {{ CodeGen: WgtNfyHandler HitInDirectoryButton
                                                                      */
static   void C_FAR RestDestWin_HitInDirectoryButton L1(
                                                    RestDestWinPtr, win)
{
  REST_UpdateDestinationSensitivity (win);
}
/* }} CodeGen: WgtNfyHandler HitInDirectoryButton
                                                                      */
```

```
/* (( CodeGen: WgtNtyHandler EltSelectedHostBox                          */
static   void C_FAR RestDestWin_EltSelectedHostBox L2(
                        RestDestWinPtr, win, CBoxEltSelectedNfyCPtr, info)
{
}
/* )) CodeGen: WgtNtyHandler EltSelectedHostBox                          */

/* (( CodeGen: WgtNfyHandler ValidateDirectoryTEd                        */
static   void C_FAR RestDestWin_ValidateDirectoryTEd L1(
                                                RestDestWinPtr, win)
{
}
/* )) CodeGen: WgtNfyHandler ValidateDirectoryTEd                        */

/* (( CodeGen: WgtNfyHandler HitBrowseButton                             */
static   void C_FAR RestDestWin_HitBrowseButton L1(
                                                RestDestWinPtr, win)
{
  REST_GetDestinationDirectory (win);
}
/* )) CodeGen: WgtNfyHandler HitBrowseButton                             */

/* (( CodeGen: WgtNfyHandler HitOKButton                                 */
static   void C_FAR RestDestWin_HitOKButton L1(RestDestWinPtr, win)
{
  Char              hostName[MAX_CLIENT_NAME_LENGTH];
                                        /* Host to restore to */
  Char              destHostName[MAX_CLIENT_NAME_LENGTH];
                                        /* Host to restore to */
  Char              dirName[MAX_STRING_LENGTH];
                                        /* Directory to restore to */
  RestoreInfoPtr    tmpInfo;
                                        /* Pointer to walk parents */

  /* If the user specified the location, update the location */
  if (!TBUT_GetSelected ((TButPtr)win->InPlaceButton))
  {

    /* Get the current destination host */
    STR_Cpy (destHostName, REST_GetCurrentDestHost(win));

    /* Get the directory from the widget */
    STR_Cpy (dirName, TED_GetStr ((TEdPtr)win->DirectoryTEd));

    /* Standardize the pathname to accept NT style paths too e.g.,
                                                        C:\XYZ */
    REST_StandardizePath(dirName);

    /*
     * Get the original host name
     */

    /* Find the client info */
    tmpInfo = currentWorkItemInfo;
    while ((tmpInfo != NULL) && (tmpInfo->type != REST_Client))
    {
      tmpInfo = tmpInfo->parent;
    }
```



```
    /* Make sure we have something */
    if (tmpInfo != NULL)
    {
      STR_Cpy (hostName, tmpInfo->name);
    }
    else
    {
      STR_Cpy (hostName, "");
    }

    /*
     * If a different host or the directory is not blank or "/"
     * verify the space available
     */
    if ((STR_Cmp (destHostName, hostName) != CMP_EQUAL) ||
        ((STR_Cmp (dirName, "") != CMP_EQUAL) &&
         (STR_Cmp (dirName, "/") != CMP_EQUAL)))
    {
      /* Verify that there is enough space to restore to */
      if (!REST_VerifySpace (win, destHostName, dirName))
      {
        return;
      }
    }
  }

  WIN_ModalReturn ((WinPtr)win, (ClientPtr)BOOL_TRUE);
}
/* )) CodeGen: WgtNfyHandler HitOKButton                                 */

/* (( CodeGen: WgtNfyHandler HitCancelButton                             */
static   void C_FAR RestDestWin_HitCancelButton L1(
                                                RestDestWinPtr, win)
{
  WIN_ModalReturn ((WinPtr)win, (ClientPtr)BOOL_FALSE);
}
/* )) CodeGen: WgtNfyHandler HitCancelButton                             */

/* (( CodeGen: WgtNfyHandler HitHelpButton                               */
static   void C_FAR RestDestWin_HitHelpButton L1(RestDestWinPtr, win)
{
  REST_DestDisplayHelp (win);
}
/* )) CodeGen: WgtNfyHandler HitHelpButton                               */

/* (( CodeGen: WgtNfyPlaceHolder ))                                      */

/* (( CodeGen: UseDefaultNfyHandler name_of_nfy_handler ))               */

/* (( CodeGen: UseAllDefaultNfyHandlers name_of_wgt_member ))            */

void      RestDestWin_Construct L1(RestDestWinPtr, win)
{
          /* ((
          CodeGen: WgtInitializations                                   */
          win ->PolicyPanel = (PanelPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "PolicyPanel");

          win->AlwaysButtone = (RButPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "AlwaysButtone");
```

```
326 1       win->OlderButton = (RButPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "OlderButton");
327 1       win->NeverButton = (RButPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "NeverButton");
328 1       win->DataPathPanel = (PanelPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "DataPathPanel");
329 1       win->NetworkButton = (RButPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "NetworkButton");
330 1       win->SymmConnButton = (RButPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "SymmConnButton");
331 1       win->LocationPanel = (PanelPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "LocationPanel");
332 1       win->InPlaceButton = (RButPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "inPlaceButton");
333 1       win->InDirectoryButton = (RButPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "InDirectoryButton");
334 1       win->HostText = (TAreaPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "HostText");
335 1       win->HosLBox = (CBoxPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "HostBox");
336 1       win->DirectoryText = (TAreaPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "DirectoryText");
337 1       win->DirectoryTEd = (STEdPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "DirectoryTEd");
338 1       win->BrowseButton = (PButPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "BrowseButton");
339 1       win->ButtonPanel = (PanelPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "ButtonPanel");
340 1       win->OKButton = (PButPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "OKButton");
341 1       win->CancelButton = (PButPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "CancelButton");
342 1       win->HelpButton = (PButPtr)PANEL_GetNamedWgt((
                                    PanelPtr)win, "HelpButton");
343 1   WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->AlwaysButtone, TBUT_NFYHIT,
344 1                   (WinWgtNfyHandlerProc)RestDestWin_HitAlwaysButtone);
345 1   WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->OlderButton, TBUT_NFYHIT,
346 1                   (WinWgtNfyHandlerProc)RestDestWin_HitOlderButton);
347 1   WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->NeverButton, TBUT_NFYHIT,
348 1                   (WinWgtNfyHandlerProc)RestDestWin_HitNeverButton);
349 1   WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->NetworkButton, TBUT_NFYHIT,
350 1                   (WinWgtNfyHandlerProc)RestDestWin_HitNetworkButton);
351 1   WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->SymmConnButton, TBUT_NFYHIT,
352 1                   (WinWgtNfyHandlerProc)RestDestWin_HitSymmConnButton);
353 1   WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->InPlaceButton, TBUT_NFYHIT,
354 1                   (WinWgtNfyHandlerProc)RestDestWin_HitInPlaceButton);
355 1   WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->InDirectoryButton, TBUT_NFYHIT,
356 1                   (
                    WinWgtNfyHandlerProc)RestDestWin_HitInDirectoryButton);
357 1   WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->HostBox, CBOX_NFYLBSELECTED,
358 1                   (WinWgtNfyHandlerProc)RestDestWin_EltSelectedHostBox);
359 1   WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->DirectoryTEd, TED_NFYVALIDATE,
360 1                   (
                    WinWgtNfyHandlerProc)RestDestWin_ValidateDirectoryTEd);
361 1   WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->BrowseButton, TBUT_NFYHIT,
362 1                   (WinWgtNfyHandlerProc)RestDestWin_HitBrowseButton);
```



```
363 1       WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->OKButton, TBUT_NFYHIT,
364 1                   (WinWgtNfyHandlerProc)RestDestWin_HitOKButton);
365 1   WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->CancelButton, TBUT_NFYHIT,
366 1                   (WinWgtNfyHandlerProc)RestDestWin_HitCancelButton);
367 1   WIN_SetWgtNfyHandler((WinPtr)win, (
                                    WgtPtr)win->HelpButton, TBUT_NFYHIT,
368 1                   (WinWgtNfyHandlerProc)RestDestWin_HitHelpButton);
369 1       /* )) CodeGen: WgtInitializations 276537
                                                                    */

371 1   WIN_SetWgtNfyHandler ((WinPtr)win,
372 1                         (WgtPtr)win,
373 1                         WIN_NFYMOUSECLICK,
374 1                         (
                          WinWgtNfyHandlerProc)GUTIL_WinHandleMouseClick);

376 1   }

378     ClientPtr RestDestWin_LoadProcess L0()
379 1   (
380 1           ClientPtr retval;
381 1           RestDestWinPtr win;

383 1           win = (RestDestWinPtr)WIN_LoadSized("restDest", "Win",
384 1                   sizeof(RestDestWinRec));
385 1           RestDestWin_Construct(win);

387 1           WIN_Init((WinPtr)win);
388 1           retval = WIN_ModalProcess((WinPtr)win);
389 1           WIN_Terminate((WinPtr)win);
390 1           return(retval);
391 1   }

393     /* )) CodeGen: WindowSection Win                                */

394     /* (( CodeGen: WindowImplementationPlaceHolder ))               */

396     /* (( CodeGen: MainPlaceHolder ))                               */

398     /*****************************************************************
399      * REST_GetDestinationInfo
400      *
401      * Description:
402      *   This routine will display the destination info dialog to get the
403      *   destination restore options from the user.
404      *
405      * Parameters:
406      *   parentWin    ( I)  - Parent window
407      *   inPlace      ( O)  - Flag for inplace restore
408      *   destHostName (IO)  - The name of the host to restore to
409      *   dirName      (
                          IO)  - The directory on the restore host to restore to
410      *   policy       ( O)  - The overwrite policy
411      *   transport    ( O)  - The restore transport method
412      *
413      * Returns:
414      *   BOOL_TRUE  - If user wants to continue with the restore
415      *   BOOL_FALSE - If user cancelled
416      *
417      *****************************************************************/

419     BoolEnum REST_GetDestinationInfo (WinPtr parentWin,
```

```
                                        BoolEnum            *inPlace,
                                        Str                 destHostName,
                                        Str                 dirName,
                                        OverwritePolicy     *policy,
                                        RestoreTransport    *transport)
{
        RestDestWinPtr win;
  BoolEnum        retVal;


        win = (RestDestWinPtr)WIN_LoadSized("restDest", "Win",
                sizeof(RestDestWinRec));
        RestDestWin_Construct(win);

  WIN_Init((WinPtr)win);

  /* Set the Window and Icon Labels */
  GUTIL_AddHostToWindowTitle ((WinPtr)win);

  /* Set the parent window */
  if (parentWin != NULL)
  {
    WIN_SetParentWin ((WinPtr)win, parentWin);
    GWin_CenterWindowInParent((WinPtr)win);
  }

  /* Set the initial state of the widgets */
  TBUT_SetSelected ((TButPtr)win->InPlaceButton, BOOL_TRUE);
  TBUT_SetSelected ((TButPtr)win->NeverButton, BOOL_TRUE);
  TED_SetStr ((TEdPtr)win->DirectoryTEd, "");

  /* Initialize to SC restore, if not available it will fix itself */
  TBUT_SetSelected ((TButPtr)win->SymmConnButton, BOOL_TRUE);

  REST_UpdateRestoreHosts (win);
  REST_UpdateDestinationSensitivity (win);
  REST_UpdateDataPath (win);

  retVal = (BoolEnum)WIN_ModalProcess((WinPtr)win);

  if (retVal)
  {
    /* Get the current destination host */
    STR_Cpy (destHostName, REST_GetCurrentDestHost(win));

    /* If the user specified the location, update the location */
    *inPlace = TBUT_GetSelected ((TButPtr)win->InPlaceButton);
    if (!*inPlace)
    {
      /* Get the directory from the widget, if allowed  */
      if (TBUT_GetSelected ((TButPtr)win->NetworkButton))
      {
        STR_Cpy (dirName, TED_GetStr ((TEdPtr)win->DirectoryTEd));

        /* Standardize the pathname to accept NT style paths too e.g.,
                                                          C:\XYZ */
        REST_StandardizePath(dirName);

        /* If the directory was left blank,
                                        start in the root directory */
        if (STR_Cmp (dirName, "") == CMP_EQUAL)
          STR_Cpy (dirName, "/");
      }
    }

    /* Get the overwrite policy */
```



```
    if (TBUT_GetSelected ((TButPtr)win->AlwaysButtone))
      *policy = Always_Overwrite;
    else if (TBUT_GetSelected ((TButPtr)win->OlderButton))
      *policy = Older_Only_Overwrite;
    else
      *policy = Never_Overwrite;

    /* Get the transport mechanism */
    if (TBUT_GetSelected ((TButPtr)win->NetworkButton))
      *transport = restoreTransportNetwork;
    else
      *transport = restoreTransportSCSI;
  }

  WIN_Terminate((WinPtr)win);

  EVENT_ProcessPending ();

  return (retVal);
}
```

```
/*****************************************************************
 * restDestMgr.c
 *
 *
 * Copyright 1999 by EMC Corp.
 *
 *
 * Mission Statement:
 *    This file contains the callback functions necessary for the
 *    EDM Restore Destination options window.
 *
 * Required Includes:
 *    None
 *
 * Compile-Time Options:
 *    N/A
 *
 *
 * RCS Information:
 *      $RCSfile$
 *      $Revision$
 *      $Date$
 *****************************************************************/

#define  ERR_LIB  RESTORE

/* Muck with the following defines to allow use of putenv which is
                                                        portable */
#define _XOPEN_SOURCE

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <errno/e_errno.h>
#include <util/esl_string.h>

#include <stdlib.h>

#include <libgen.h>
#include <time.h>

#include <tedpub.h>
#include <winpub.h>
#include <stripub.h>

#include <restore/restore_api.h>
#include "restDest.h"
#include "restore.h"
#include "restoreP.h"
#include "restore/restMgr.h"
#include "restUtils.h"
#include "gutil/iconUtils.h"
#include "gutil/hostUtils.h"
#include "gutil/winUtils.h"
#include "gutil/cboxUtils.h"
#include "gutil/resUtils.h"
#include "gutil/miscUtils.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "gutil/alertMgr.h"
#include "help/helpDefs.h"
#include "help/helpIpc.h"
#include "gutil/fBrowseMgr.h"

ERR_EXTERN
ERR_!NMODULE("restore")
```



```
/*************
 * Constants *
 *************/

/*************************
 * Local Global Variables *
 *************************/

/*****************************************************************
 * REST_UpdateDataPath
 *
 * Description:
 *   This routine will update the data path visibility base on what is
 *   allowed for the current work item.
 *
 * Parameters:
 *   None
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_UpdateDataPath (RestDestWinPtr win)
{
  BoolEnum  isSymmOK=TRUE;
  BoolEnum  isNetworkOK=TRUE;
  eerrno_ty status;

  /* Based on the current WI determin if SC restore is OK */
  if ((currentWorkItemInfo != NULL) &&
      (currentWorkItemInfo->restoreObject != NULL))
  {
    /* Make sure the current work item is restorable over SymmConnect
                                                              */
    if ((status = EDMRST_GetSymmRestoreOption(GREST_Handle,
                                        currentWorkItemInfo->restoreObject,
                                                  &isSymmOK)) != E_SUCCESS)
    {
      REST_DisplayErrnoMessage ((WinPtr)win, NULL, NULL, status);

      /* on error, allow SC restore */
      isSymmOK = BOOL_TRUE;
    }

    /* Make sure the current work item is restorable over the network
                                                              */
    if ((status = EDMRST_GetNetworkRestoreOption(GREST_Handle,
                                        currentWorkItemInfo->restoreObject,
                                                  &isNetworkOK)) !=
                                                          E_SUCCESS)
    {
      REST_DisplayErrnoMessage ((WinPtr)win, NULL, NULL, status);

      /* on error, allow network restore */
      isNetworkOK = BOOL_TRUE;
    }
  }
  else
  {
    /*
     * Allow SC and Network to be visible
     * they will be updated more accurately later
```

```
127 4       */
128 2
129 2       isSymmOK = BOOL_TRUE;
130 2       isNetworkOK = BOOL_TRUE;
131 1     }
132 1
133 1     if (!isNetworkOK)
134 2     {
135 2       /* Not OK to use network, so select the SymmConnect button */
136 2       TBUT_Select ((TButPtr)win->SymmConnButton);
137 1     }
138 1
139 1     if (!isSymmOK)
140 2     {
141 3       /* Not OK to use symm connect, so select the network button */
142 2       TBUT_Select ((TButPtr)win->NetworkButton);
143 1     }
144 1
145 1     /* Set the visibility of the symm connect button */
146 1     GUTIL_WGT_SetEnabled ((WgtPtr)win->SymmConnButton, isSymmOK);
147 1
148 1     /* Set the visibility of the network button */
149 1     GUTIL_WGT_SetEnabled ((WgtPtr)win->NetworkButton, isNetworkOK);
150 1
151 1     /* Update the destination sensitivity */
152 1     REST_UpdateDestinationSensitivity (win);
153 1   }
154
155   /*****************************************************************
156    * REST_GetCurrentDestHost
157    *
158    * Description:
159    *   This routine will determin the current destination host.
                                                          If the user
160    *   has selected a host it will return that host, otherwise if will
161    *   return the current source host if the user is working on one,
                                                          otherwise
162    *   it will simply return the local host.
163    *
164    * Parameters:
165    *   None.
166    *
167    * Returns:
168    *   None.
169    *
170    *****************************************************************/
171
172   Str REST_GetCurrentDestHost (RestDestWinPtr win)
173   {
174     static Char     destHostName[MAX_CLIENT_NAME_LENGTH];
                                                  /* Current dest host */
175     RestoreInfoPtr tmpInfo;
                                                  /* Pointer to walk list */
176
177     /* Get the current selected host, if the user has chosen one */
178     if (CBOX_HasChoice (win->HostBox))
179     {
180       STR_Cpy (destHostName, CBOX_ChosenGetLabel(win->HostBox));
181     }
182     else
183     {
184       /*
185        * Get the current source client
186        */
187
188       /* Use the current info */
```



```
189 2     if (currentWorkItemInfo != NULL)
190 2       tmpInfo = currentWorkItemInfo;
191 2     else
192 2       tmpInfo = NULL;
193
194 2     /* Find the client for this restoral */
195 2     while ((tmpInfo != NULL) && (tmpInfo->type != REST_Client))
196 3     {
197 3       tmpInfo = tmpInfo->parent;
198 2     }
199
200 2     /* If we found the client get its name */
201 2     if (tmpInfo != NULL)
202 3     {
203 3       STR_Cpy (destHostName, tmpInfo->name);
204 2     }
205 2     else
206 3     {
207 3       /* Didn't find it, use the current host */
208 3       STR_Cpy (destHostName, GUTIL_GetThisHost());
209 2     }
210 1   }
211
212 1   return (destHostName);
213   }
214
215   /*****************************************************************
216    * REST_UpdateBrowseButtonSensitivity
217    *
218    * Description:
219    *   This routine sets the sensitivity of the browse button.
220    *
221    *   The browse button is sensitive only if:
222    *     1) The specify location toggle is set
223    *     2) The destination host supports browsing
224    *          (currently Solaris, HP/UX and AIX)
225    *
226    * Parameters:
227    *   None.
228    *
229    * Returns:
230    *   None.
231    *
232    *****************************************************************/
233
234   void REST_UpdateBrowseButtonSensitivity (RestDestWinPtr win)
235   {
236     Char      destHostName[MAX_CLIENT_NAME_LENGTH];
                                          /* Host to restore to */
237     BoolEnum            sensitive;
                                          /* Flag if button should be sensitive */
238     BuCfgPlatformType platformType;
                                          /* Platform for destination host */
239     eerrno_ty           status;
240
241     /* If there is a restore in progress, don't update anything */
242     if (REST_RestoreInProgress ())
243       return;
244
245     /* Check if specific location is on */
246     if (TBUT_GetSelected((TButPtr)win->InDirectoryButton) &&
247         TBUT_GetSelected ((TButPtr)win->NetworkButton))
248     {
249       /* Get the current host */
250       STR_Cpy (destHostName, REST_GetCurrentDestHost (win));
```

```
252 2     /* Get the platform type for the host */
253 2     if ((status = EDMRST_GetHostPlatformType (GREST_Handle,
254 2                                               destHostName,
255 2                                               &platformType)) ==
                                                                 E_SUCCESS)
256 3     {
257 3       /* Can only browse UNIX or NT clients */
258 3       if ((platformType == BUCFG_PLATFORM_UNIX) ||
259 3           (platformType == BUCFG_PLATFORM_WNT))
260 4       {
261 4         sensitive = BOOL_TRUE;
262 4       }
263 3       /* All others, don't allow browsing */
264 4       else
265 4       {
266 4         sensitive = BOOL_FALSE;
267 4       }
268 2     }
269 2     /* Unable to get platform type */
270 2     else
271 3     {
272 3       REST_DisplayErrnoMessage ((WinPtr)win, NULL, NULL, status);

274 3       /* Assume we can browse,
                                    browser will tell us later if we can't */
275 3       sensitive = BOOL_TRUE;
276 2     }
277 1   }

279 1   /* This is an inplace restore, no browsing necessary */
280 1   else
281 2   {
282 2     sensitive = BOOL_FALSE;
283 1   }

285 1   GUTIL_WGT_SetEnabled ((WgtPtr)win->BrowseButton,  sensitive);
286   }

288   /*****************************************************************
289    * REST_UpdateDestinationSensitivity
290    *
291    * Description:
292    *   This routine set the sensitivity of the user chosen destination
293    *   widgets.  If the user has selected in place, then disable the
294    *   destination options, otherwise enable them.
295    *
296    * Parameters:
297    *   None.
298    *
299    * Returns:
300    *   None.
301    *
302    *****************************************************************/

304   void  REST_UpdateDestinationSensitivity (RestDestWinPtr win)
305 1 {
306 1   BooleEnum isDestinationSelectable;
                                          /* Is the destination selectable */

308 1   /* If there is a restore in progress, don't update anything */
309 1   if (REST_RestoreInProgress ())
310 1     return;

312 1   /* Check if the user in using in place or not */
313 1   isDestinationSelectable = !TBUT_GetSelected((
                                                TButPtr)win->InPlaceButton);
```



```
315 1   /* Enable or disable the widgets appropriately */
316 1   GUTIL_WGT_SetEnabled ((WgtPtr)win->HostText,
317 1                         isDestinationSelectable);
318 1   GUTIL_WGT_SetEnabled ((WgtPtr)win->HostBox,
319 1                         isDestinationSelectable);

321 1   /* If doing a cross restore,
                                  enable directory tedd only if not Symm Connect */
322 1   if (TBUT_GetSelected ((TButPtr)win->NetworkButton))
323 2   {
324 2     GUTIL_WGT_SetEnabled ((WgtPtr)win->DirectoryText,
325 2                           isDestinationSelectable);
326 2     GUTIL_WGT_SetEnabled ((WgtPtr)win->DirectoryTEd,
327 2                           isDestinationSelectable);

329 2     /* For Network restores, allow overwrite policy */
330 2     if (!WGT_IsEnabled ((WgtPtr)win->OlderButton))
331 3     {
332 3       TBUT_Select ((TButPtr)win->NeverButton);
333 3       GUTIL_WGT_SetEnabled ((WgtPtr)win->OlderButton, BOOL_TRUE);
334 3       GUTIL_WGT_SetEnabled ((WgtPtr)win->NeverButton, BOOL_TRUE);
335 2     }
336 1   }
337 1   else
338 2   {
339 2     GUTIL_WGT_SetEnabled ((WgtPtr)win->DirectoryText, BOOL_FALSE);
340 2     GUTIL_WGT_SetEnabled ((WgtPtr)win->DirectoryTEd, BOOL_FALSE);

342 2     /* For SC restores, do NOT allow overwrite policy */
343 2     if (WGT_IsEnabled ((WgtPtr)win->OlderButton))
344 3     {
345 3       TBUT_Select ((TButPtr)win->AlwaysButton);
346 3       GUTIL_WGT_SetEnabled ((WgtPtr)win->OlderButton, BOOL_FALSE);
347 3       GUTIL_WGT_SetEnabled ((WgtPtr)win->NeverButton, BOOL_FALSE);
348 2     }
349 1   }

351 1   /* Check if browsing should be enabled */
352 1   REST_UpdateBrowseButtonSensitivity (win);
353   }

355   /*****************************************************************
356    * REST_UpdateRestoreHosts
357    *
358    * Description:
359    *   This routine will update the restore destination box to the
                                                              current
360    *   choices available.
361    *
362    * Parameters:
363    *   info (I) - Child info record
364    *
365    * Returns:
366    *   None.
367    *
368    *****************************************************************/

370   void  C_FAR  REST_UpdateRestoreHosts (RestDestWinPtr win)
371 1 {
372 1   RestoreInfoPtr  tmpInfo;
                                          /* Pointer to walk parents */
373 1   char            *hosts[HOSTS_BUFFER_LENGTH]; /* Hosts returned */
374 1   long             cookie = INIT_COOKIE;
                                          /* Ah, the magic cookie */
375 1   short           numEntries;
```

```
                                                    /* Number of hosts returned */
    int             i;                              /* Loop Counter */
    errno_t         eerrno;                         /* Error status */


    /* Find the client info */
    tmpInfo = currentWorkItemInfo;
    while ((tmpInfo != NULL) && (tmpInfo->type != REST_Client))
    {
      tmpInfo = tmpInfo->parent;
    }

    /* Make sure we have something */
    if (tmpInfo != NULL)
    {

      /* First, clear out any old hosts: */
      CBOX_GoFirst(win->HostBox);
      while (CBOX_IsOk (win->HostBox))
      {
        CBOX_GoFirst (win->HostBox);
        CBOX_CurRemoveElt (win->HostBox);
      }

      /* Allocate temporary storage for the hosts to be returned */
      for (i = 0; i < HOSTS_BUFFER_LENGTH; i++)
        hosts[i] = (char *) GUTIL_Malloc (
                                    MAX_CLIENT_NAME_LENGTH * sizeof(char));

      /* Get all the templates for the new work item */
      while (cookie != DONE_COOKIE)
      {
        numEntries = 0;
        if ((eerrno = EDMRST_GetDestinationHosts (GREST_Handle,
                                                  HOSTS_BUFFER_LENGTH,
                                                  hosts,
                                                  &numEntries,
                                                  &cookie)) ==
                                                              E_SUCCESS)
        {

          /* Add all the hosts to the list */
          for (i=0; i<numEntries; i++)
          {
            /* Add this host to the Combo Box, sorted */
            CBOX_AddStrToCbox (win->HostBox, hosts[i]);

            /* By default, select the work-item's source host */
            if (STR_Cmp (tmpInfo->name, hosts[i]) == CMP_EQUAL)
            {
              CBOX_CurSelect (win->HostBox);
            }
          }

        }
        else
        {
          REST_DisplayErrnoMessage ((WinPtr)win, NULL, NULL, eerrno);

          /* Just get out */
          cookie = DONE_COOKIE;
        }
      }

      /* Free up the temporary storage */
      for (i = 0, i < HOSTS_BUFFER_LENGTH; i++)
        GUTIL_Free (hosts[i]);
```



```
    }
  }

  /*************************************************************
   * REST_GetDestinationDirectory
   *
   * Description:
   *   This routine will display the filesystem browser window for the
   *   current destination host.  If the user selects a directory from
   *   the browser,
                    this routine will then update the current destination
   *   directory.
   *
   * Parameters:
   *   None.
   *
   * Returns:
   *   None.
   *
   *************************************************************/

  void       REST_GetDestinationDirectory (RestDestWinPtr win)
  {
    Char             destHostName[MAX_CLIENT_NAME_LENGTH];
                                                  /* Host to restore to */
    Char            *dirName;
                                                  /* New Directory */
    Char             currentDirectory[MAX_STRING_LENGTH];
                                                  /* Current Directory */


    /* Get the current destination host */
    STR_Cpy (destHostName, REST_GetCurrentDestHost (win));

    /* Get the current directory from the widget */
    STR_Cpy (currentDirectory, TED_GetStr
                 ((TEdPtr)win->DirectoryTEd));

    /* Let the user choose a new directory via the browser */
    dirName = GFBROWS_DisplayWin ((WinPtr)win,
                                  "Select Restore Destination Directory",
                                  destHostName,
                                  currentDirectory,
                                  NULL,
                                  GFBROWS_HIDE_FILES,
                                  REST_MOD1D,
                                  REST_DESTINATION_BROWSE);


    /* If the user chose a new directory, set the directory widget */
    if (dirName != NULL)
    {
      TED_SetStr ((TEdPtr)win->DirectoryTEd, dirName);
      GUTIL_Free (dirName);
    }

  }

  /*************************************************************
   * REST_DestDisplayHelp
   *
   * Description:
   *   This routine will display help for the destination window
   *
   * Parameters:
   *   None.
   *
   * Returns:
```

```
501     *    None.
502     *
503     ************************************************************/
504
505     void REST_DestDisplayHelp (RestDestWinPtr win)
506 1   {
507 1     EDMHELP_Display ((WinPtr)win, REST_MODID, REST_DESTINATION_OPTIONS);
508     }
```

```
/*****************************************************************
 * restFileMgr.c
 *
 *
 * Copyright 1996 by Epoch Systems, Inc.
 *
 *
 * Mission Statement:
 *    This file contains the functions necessary for the file manager
 *    portion of the EDM Restore window.
 *
 * Required Includes:
 *    None
 *
 * Compile-Time Options:
 *    N/A
 *
 *
 * RCS Information:
 *      $RCSfile$
 *      $Revision$
 *      $Date$
 *****************************************************************/

#define  ERR_LIB  RESTORE

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>

#include <stdlib.h>

#include <libgen.h>
#include <time.h>

#include <apppub.h>
#include <eventpub.h>
#include <rlibpub.h>
#include <gwpub.h>
#include <respub.h>
#include <lboxpub.h>
#include <mbarpub.h>
#include <panelpub.h>
#include <tareapub.h>
#include <tbutpub.h>
#include <tedpub.h>
#include <winpub.h>
#include <strlpub.h>
#include <drawpub.h>
#include <dsplypub.h>
#include <ibutpub.h>
#include <arraypub.h>

#include "werrno/e_errno.h"
#include "util/esl_string.h"
#include <restore/restore_api.h>
#include "restore/restMgr.h"
#include "restore.h"
#include "restore2.h"
#include "restCBMgr.h"
#define REST_FILE_INIT
#include "restFileMgr.h"
#undef REST_FILE_INIT
#include "restCalendar.h"
#include "restSearch.h"
#include "restSelMgr.h"
#include "restUtils.h"
```



```
#include "restAPIUtils.h"
#include "gutil/timeUtils.h"
#include "gutil/iconUtils.h"
#include "gutil/winUtils.h"
#include "gutil/resUtils.h"
#include "gutil/miscUtils.h"
#include "gutil/fileMgr.h"
#include "gutil/tabDefs.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "help/helpDefs.h"
#include "help/helpIpc.h"
#include "gutil/codeTracer.h"
#include "gutil/alertMgr.h"
#include "ipc/ipcl.h"

ERR_EXTERN
ERR_INMODULE("restore")

/*************
 * Constants *
 *************/

#define NUMBER_ITEMS_COLUMNS    9

/*************************
 * Local Data Structures *
 *************************/

/*************************
 * Local Global Variables *
 *************************/

/* The file manager context for the restore file manager */
static GFMGR_Context  fileMgrContext;

/* Flags for current state of processing */
static BoolEnum  allowGetChildren = BOOL_TRUE;
static BoolEnum  reReadInProgress = BOOL_FALSE;

/*****************************************************************
 * REST_GetFMgrContext
 *
 * Description:
 *    This routine will return the current file manage context for the
 *    restore window.
 *
 * Parameters:
 *    None.
 *
 * Returns:
 *    None.
 *
 *****************************************************************/

GFMGR_Context REST_GetFMgrContext (void)
{
  return (fileMgrContext);
}
```

```
127  /*****************************************************************
128   * REST_IsReReadInProgress
129   *
130   * Description:
131   *   This routine will determine if a re-read of a work-item is
                                                                currently
132   *   in progress.
133   *
134   * Parameters:
135   *   None.
136   *
137   * Returns:
138   *   BOOL_TRUE  - If there is a re-read in progress.
139   *   BOOL_FALSE - otherwise
140   *
141   *****************************************************************/

143  BoolEnum  REST_IsReReadInProgress (void)
144  {
145    return (reReadInProgress);
146  }
```



```
148  /*****************************************************************
149   * REST_SetReReadInProgress
150   *
151   * Description:
152   *   This routine will set the re-read in progress status to the given
153   *   status.
154   *
155   * Parameters:
156   *   status (I) - Flag if there is a re-read in progress
157   *
158   * Returns:
159   *   None.
160   *
161   *****************************************************************/

163  void  REST_SetReReadInProgress (BoolEnum status)
164  {
165    reReadInProgress = status;
166  }
```

```
168   /*****************************************************************
169    * REST_ReReadWorkItem
170    *
171    * Description:
172    *    This routine will re-read all restorable objects for a given
173    *    work item.  It frees all the current children of the work item
174    *    then uses the file manager to re-get the children.  This routine
175    *    attempts to keep the current file manager selection as it was.
176    *    but this is not always possible.
177    *
178    * Parameters:
179    *    currentWI (I) - The work-item info to re-read.
180    *
181    * Returns:
182    *    None.
183    *
184    *****************************************************************/

186   void  REST_ReReadWorkItem (RestoreInfoPtr currentWI)
187 1 {
188 1   RestoreInfoPtr tmpInfo;            /* Temporary info pointer */
189 1   RestoreInfoPtr nextInfo;           /* Pointer to next info */
190 1   RestoreInfoPtr foundInfo = NULL;  /* Info found for selected item */

192 1   /* Free all the current children */
193 1   tmpInfo = currentWI->children;
194 1   while (tmpInfo != NULL)
195 2   {
196 2     nextInfo = tmpInfo->next;
197 2     REST_FreeInfo (tmpInfo);
198 2     tmpInfo = nextInfo;
199 1   }
200 1   currentWI->children = NULL;

202 1   /* Flag that a re-read is in progress */
203 1   REST_SetReReadInProgress (BOOL_TRUE);

205 1   /* Call the file manager to re-get the new children */
206 1   GFMGR_FreezeDisplay (fileMgrContext);

208 1   /* Update the file manager children */
209 1   GFMGR_UpdateChildren (fileMgrContext, currentWI);

211 1   /* Don't allow any reading of children,
                                        only use what we currently have  */
212 1   allowGetChildren = BOOL_FALSE;

214 1   /* If we have any children, try to find the selected child */
215 1   if (currentWI->children != NULL)
216 2   {
217 2     /* Select the object which corresponds to the currently selected
                                                          name */
218 2     foundInfo = REST_FindSelectionString ();
219 1   }
220 1   else
221 1   {
222 2     /* Just use the workitem as the selected item */
223 2     foundInfo = currentWI;
224 1   }


227 1   /* Now select the object */
228 1   if (foundInfo != NULL)
229 2   {
230 2     GFMGR_SelectObject (fileMgrContext, (
```



```
                                          GFMGR_Object)foundInfo, BOOL_FALSE);
231 2     REST_SetSelectionString (REST_GetFullName(foundInfo));
232 1   }
233 1   else
234 2   {
235 2     GFMGR_DeselectAllObjects (fileMgrContext, BOOL_FALSE);
236 1   }

238 1   /* Update the file manager display */
239 1   GFMGR_Display (fileMgrContext);

241 1   /* Flag that the re-read is in done */
242 1   REST_SetReReadInProgress (BOOL_FALSE);
243 1   allowGetChildren = BOOL_TRUE;
244   }
```

```
/*****************************************************************
 * REST_GetBackupCellString
 *
 * Description:
 *   This routine will return the string to display in the given cell
 *   for the given object.
 *
 * Parameters:
 *   object          (I) - The backup object.
 *   column          (I) - The column to get the string for.
 *   returnString    (O) - The string to display.
 *   justification (O) - The justification to use.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void  REST_GetBackupCellString (GFMGR_Context fMgr,
                                GFMGR_Object  object,
                                Int           column,
                                Str           returnString,
                                UInt16        *justification)
{
  RestoreInfoPtr   info; /* The info record for the object */

  /* Convert the given object to its real type */
  info = (RestoreInfoPtr)object;

  /* Based on the column, get the string */
  switch (column)
  {
    case 4:
      /* If it is a work item draw nothing */
      if (info->type == REST_WorkItem)
      {
        STR_Cpy (returnString, "");
      }
      /* If it is a failed workitem display the error string */
      else if ((info->type == REST_FailedWorkItem) && (
                                            info->errorString != NULL))
      {
        STR_Cpy (returnString, info->errorString);
      }
      /* If it is an error object draw nothing */
      else if (info->type == REST_ErrorObject)
      {
        STR_Cpy (returnString, "");
      }
      /* else draw the accoss info string */
      else
      {
        STR_Cpy (returnString, REST_GetPermString (info));
      }
      *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;
      break;

    case 5:
      /* If it is a work item draw nothing */
      if (info->type == REST_WorkItem)
      {
        STR_Cpy (returnString, "");
      }
      /* If it is a failed workitem draw nothing */
      else if (info->type == REST_FailedWorkItem)
      {
        STR_Cpy (returnString, "");
      }
      /* else draw the owner string */
      else
      {
        STR_Cpy (returnString, REST_GetOwnerString (info));
      }
      *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;
      break;

    case 6:
      /* If it is a work item or a database object draw nothing */
      if (info->type == REST_WorkItem)
      {
        STR_Cpy (returnString, "");
      }
      /* If it is a failed workitem draw nothing */
      else if (info->type == REST_FailedWorkItem)
      {
        STR_Cpy (returnString, "");
      }
      /* If it is an error object  draw nothing */
      else if (info->type == REST_ErrorObject)
      {
        STR_Cpy (returnString, "");
      }
      /* else draw the group string */
      else
      {
        STR_Cpy (returnString, REST_GetGroupString (info));
      }
      *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;
      break;

    case 7:
      /* If it is a work item or a database object draw nothing */
      if (info->type == REST_WorkItem)
      {
        STR_Cpy (returnString, "");
      }
      /* If it is a failed workitem draw nothing */
      else if (info->type == REST_FailedWorkItem)
      {
        STR_Cpy (returnString, "");
      }
      /* If it is an error object  draw nothing */
      else if (info->type == REST_ErrorObject)
      {
        STR_Cpy (returnString, "");
      }
      /* else draw the size string */
      else
      {
        STR_Cpy (returnString, REST_GetSizeString (info));
      }
      *justification = DRAW_JUSTRIGHT | DRAW_JUSTVCENTER;
      break;

    case 8:
      /* If it is a work item or a database object draw nothing */
```

```
376 2       if (info->type == REST_WorkItem)
377 3       {
378 3         STR_Cpy (returnString, "");
379 2       }
380 2       /* If it is a failed workitem draw nothing */
381 2       else if (info->type == REST_FailedWorkItem)
382 3       {
383 3         STR_Cpy (returnString, "");
384 2       }
385 2       /* if it is an error object  draw nothing */
386 2       else if (info->type == REST_ErrorObject)
387 3       {
388 3         STR_Cpy (returnString, "");
389 2       }
390 2       /* else draw the date string */
391 2       else
392 3       {
393 3         STR_Cpy (returnString, REST_GetDateString (info));
394 2       }
395 2       *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;
396 2       break;

398 2     case 9:
399 2       /* If it is a work item or a database object draw nothing */
400 2       if (info->type == REST_WorkItem)
401 3       {
402 3         STR_Cpy (returnString, "");
403 2       }
404 2       /* If it is a failed workitem draw nothing */
405 2       else if (info->type == REST_FailedWorkItem)
406 3       {
407 3         STR_Cpy (returnString, "");
408 2       }
409 2       /* If it is an error object  draw nothing */
410 2       else if (info->type == REST_ErrorObject)
411 3       {
412 3         STR_Cpy (returnString, "");
413 2       }
414 2       /* else draw the time string */
415 2       else
416 3       {
417 3         STR_Cpy (returnString, REST_GetTimeString (info));
418 2       }
419 2       *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;
420 2       break;

422 2     default:
423 2       STR_Cpy (returnString, "");
424 2       *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;
425 2       break;
426 1   }

428   }
```



```
430   /*****************************************************************
431    * REST_ShowPath
432    *
433    * Description:
434    *   This routine will display the full path name for the given object
435    *
436    * Parameters:
437    *   fullPath (I) - The full path to display.
438    *
439    * Returns:
440    *   None.
441    *
442    *****************************************************************/

444   void REST_ShowPath (Str fullPath)
445 1 {
446 1   /* Set the path widget to the given full path */
447 1   TED_SetStr ((TEdPtr)REST_RestoreWin->PathTEd, fullPath);
448   }
```

```
/*****************************************************************
 * REST_ShowObject
 *
 * Description:
 *   This routine will display the given object in the file manager
 *   and select it.
 *
 * Parameters:
 *   object (I) - The Object to display.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_ShowObject (RestoreInfoPtr object)
{

  /* Validate the object */
  if (object != NULL)
  {
    /* First show the parent object */
    if (object->parent != NULL)
    {
      REST_ShowObject (object->parent);

      /* Open the parent */
      GFMGR_OpenObject (fileMgrContext, object->parent);
    }

    /* Now select this object */
    GFMGR_SelectObject (fileMgrContext, (
                                        GFMGR_Object)object, BOOL_FALSE);
    REST_SetSelectionString (REST_GetFullName(object));

    /* Set the current path to the selected item */
    if ((object->type == REST_Client) || (
                                        object->type == REST_WorkItem)
      REST_ShowPath ("");
    else
      REST_ShowPath (REST_GetFullName(object));
  }
}
```



```
/*****************************************************************
 * REST_SetBackupView
 *
 * Description:
 *   This routine will display the object with the given full name
 *   at the given backup time.
 *
 * Parameters:
 *   backupTime   (I) - The time for the backup to display
 *   itemFullName (I) - The full name of the object to display
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void  REST_SetBackupView (time_t    backupTime,
                          Str       itemFullName)
{
  RestoreInfoPtr  info;
                                    /* The info for the given name */
  RestoreInfoPtr  selectedObject;   /* Currently selected object */
  eerrno_ty       eerrno;           /* Error status */
  time_t          currentBackupTime;  /* Time of the current backup */
  u_long          flags;

  if (TBUT_GetSelected ((TButPtr)REST_RestoreWin->AllowPartialButton))
    flags = BACKUP_SELECTION_FLAG_PARTIAL_OK;
  else
    flags = BACKUP_SELECTION_FLAG_COMPLETE_ONLY;

  /* Get the current backup time */
  if ((eerrno = EDMRST_GetCurrentBackupTime (GREST_Handle,
                                        &currentBackupTime)) !=
                                                              0)
  {
    /* Hmm... I guess we just use time zero */
    currentBackupTime = 0;
  }

  /* If the backup time is different then the current backup time */
  if ((backupTime != 0) && (backupTime != currentBackupTime))
  {

    /* Set the backup to the given time */
    if ((eerrno = EDMRST_SetBackupForTime(
                                        GREST_Handle, backupTime, flags))== 0)

    {

      /* Update the date */
      if (currentWorkItemInfo != NULL)
      {
        REST_UpdateBackupDate ();
      }
    }
    else
    {
      Char        outputString[2 * GMAX_OBJECT_LENGTH];
                                              /* String to display */

      /* couldn't switch, create the error message */
      STR_Sprintf (outputString,
                   REST_GetErrorString (
                                        REST_BACKUP_TIME_SWITCH_ERROR),
                   REST_GetTimeDateString (backupTime));
```

```
553  3        /* Display the error message */
554  3        REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
555  3                                   NULL,
556  3                                   outputString,
557  3                                   eerrno);

559  3        /* We're done here */
560  3        return;
561  2      }
562  1    }

564  1    /* Try to find the object in the current work item */
565  1    info = REST_FindInfoInChildren (
                                  itemFullName, currentWorkItemInfo, BOOL_TRUE);

567  1    /* If we found it */
568  1    if (info != NULL)
569  2    {
570  2      /* If this is a file object,
                                  select its parent to show object in LBOX */
571  2      if ((info->type == REST_File) && (info->parent != NULL))
572  3      {
573  3        selectedObject = info->parent;
574  2      }

576  2      /* Else select this object */
577  2      else
578  3      {
579  3        selectedObject = info;
580  2      }

582  2      /* Show the object in the file manager if not already selected */
583  2      if (!GFMGR_IsObjectSelected (fileMgrContext, (
                                              GFMGR_Object)selectedObject))
584  3      {
585  3        GFMGR_FreezeDisplay (fileMgrContext);
586  3        REST_ShowObject (selectedObject);
587  3        GFMGR_Display (fileMgrContext);

589  3        /* Update the current backup options to the new selected object
                                                                        */
590  3        REST_UpdateBackupOptions (info);
591  2      }

593  2      /* IF the object is not a parent, select it in the list box */
594  2      if (selectedObject != info)
595  2        GFMGR_SelectObjectInLBox (fileMgrContext, (
                                          GFMGR_Object)info, BOOL_TRUE);
596  1    }
597  1    else
598  2    {
599  2      Char        outputString[4 * GMAX_OBJECT_LENGTH];
                                                  /* Error string to show */

601  2      /* Create the error string */
602  2      STR_Sprintf (outputString,
603  2                   REST_GetErrorString (REST_SET_VIEW_ERROR),
604  2                   itemFullName);

606  2      /* Display the error message */
607  2      GALERT_DisplayError ((WinPtr)REST_RestoreWin,
608  2                           REST_GetErrorString (REST_ERROR_INDEX),
609  2                           GICON_GetError(),
610  2                           outputString);
611  1    }
```



```
613  1  }
```

```
615  /*****************************************************************
616   * REST_SetViewToPath
617   *
618   * Description:
619   *   This routine will display the object with the full name currently
620   *   in the path widget in the current backup time.
621   *
622   * Parameters:
623   *   None.
624   *
625   * Returns:
626   *   None.
627   *
628   *****************************************************************/
629
630  void  REST_SetViewToPath (void)
631  {
632    Str       itemFullName;
                                        /* The full path to set the view to */
633    BoolEnum lastCharFound = BOOL_FALSE;
                                        /* Flag if we verified last char */
634    Int       lastChar;
                                        /* Current position of last char */
635
636    /* Get the current path */
637    itemFullName = osl_strdup ((Str)TED_GetStr((
                                        TEdPtr)REST_RestoreWin->PathTEd));
638
639    /* Strip off trailing spaces and directory markers */
640    REST_StripDirectoryChars (itemFullName);
641
642    /* Set the backup view to this item at the current time */
643    REST_SetBackupView (0, itemFullName);
644
645    /* We're done with the name so free it */
646    GUTIL_Free (itemFullName);
647  }
```



```
648  /*****************************************************************
649   * REST_GetChildren
650   *
651   * Description:
652   *   This routine will return the children of the parent object via
653   *   the AddChild routine.
654   *
655   * Parameters:
656   *   parent      (I) - The parent to get the children for.
657   *   isSelected (I) - Flag whether or not the object is selected
658   *
659   * Returns:
660   *   None.
661   *
662   *****************************************************************/
663
664  void REST_GetChildren (GFMGR_Context fMgr,
665                         GFMGR_Object  parent,
666                         BoolEnum      isSelected)
667  {
668    RestoreInfoPtr  info;
                                        /* The info for the given parent */
669    RestoreInfoPtr  tmpObject;
                                        /* Pointer to walk the children with */
670    RestoreInfoPtr  originalWI;
                                        /* Save pointer to original work item */
671    BoolEnum        found = BOOL_FALSE;
                                        /* Flag whether or not we found it */
672    BoolEnum        newWI = BOOL_FALSE;
                                        /* Flag whether or not we have a new WI */
673    Int             count = 0;
                                        /* Number of children added */
674
675    /* Convert to the real data type */
676    info = (RestoreInfoPtr)parent;
677
678    /* If the info is NULL or it is a file, no children to add */
679    if ((info == NULL) || (info->type == REST_File))
680      return;
681
682    /* If this object is currently selected,
683                                        see if the work-item has changed */
684    if ((isSelected) &&
685        (!REST_IsReReadInProgress()) &&
686        (currentWorkItemInfo != NULL) &&
687        (info->type != REST_Client))
688    {
689
690      /* Keep a pointer to the original work item */
691      originalWI = currentWorkItemInfo;
692
693      /* Find work-item for parent object */
694      tmpObject = info;
695      while (!(found) && (tmpObject != NULL))
696      {
697        /* Is this object the work-item */
698        if (tmpObject->type == REST_WorkItem)
699        {
700          /* If this is not the current work-item, make is so */
701          if (currentWorkItemInfo != tmpObject)
702          {
703            currentWorkItemInfo = tmpObject;
704            newWI = BOOL_TRUE;
705          }
706          found = BOOL_TRUE;
707        }
```

```
708 1       else
709 4       {
710 6         /* Go to the parent of this object */
711 4         tmpObject = tmpObject->parent;
712 3       }
713 2     }
714 1   }

716 1   /* If this is a different work-item, re-read the work-item data */
717 1   if ((newWI) && (allowGetChildren))
718 2   {
719 2     REST_HaReadWorkItem (currentWorkItemInfo);

721 2     /* Clear the mark flags for all objects */
722 2     REST_ClearObjectMarks (originalWI);
723 1   }
724 1   else
725 2   {

727 2     /*  Create the children if need be, and this is selected */
728 2     if ((info->children == NULL) && (allowGetChildren))
729 3     {
730 3       REST_CreateInfoChildren (info);

732 2     }

734 2     /* Add the children to the file manager */
735 3     tmpObject = info->children;
736 2     while (tmpObject != NULL)
737 3     {

739 3       /*
740 3        * Check if this object should be shown in the file manager
741 3        *   Verify hidden file
742 3        *   Verify offsite file
743 3        *   Verify bad file
744 3        */

746 3       if ((REST_ShowHiddenFiles || (tmpObject->name[0] != '.')) &&
747 3           (REST_ShowBadFiles || !REST_IsBadObject (tmpObject)))
748 4       {
749 4         count++;
750 4         GFMGR_AddChild (fileMgrContext, parent, (
                                                    GFMGR_Object)tmpObject);
751 3       }
752 3       tmpObject = tmpObject->next;
753 2     }
754 1   }

756 1   /* If this is a workitem, update the backup options */
757 1   if ((info->type == REST_WorkItem) &&
758 1       (allowGetChildren) &&
759 1       (isSelected) &&
760 1       (info->children != NULL))
761 2   {
762 2     /* Set the WorkItem options to visible */
763 2     REST_SetFSOptionsVisibility (BOOL_TRUE);
764 2     REST_SetSearchVisiblity (BOOL_TRUE);

766 2     /* Update the partial backup button availability */
767 2     REST_UpdatePartialButton ();

769 2     /* Update the workitem data widgets */
770 2     REST_UpdateBackupTemplates (info);
771 1   }
```



```
773   }
```

```
775  /*****************************************************************
776   * REST_CloseChildren
777   *
778   * Description:
779   *   This routine is provided for the file manager to call when an
780   *   object is closed.  It does nothing.
781   *
782   * Parameters:
783   *   parent      (I) - The parent to close.
784   *
785   * Returns:
786   *   None.
787   *
788   *****************************************************************/
789
790  void REST_CloseChildren (GFMGR_Context fMgr, GFMGR_Object parent)
791  {
792    /* For Speed,
793             don't destroy anything until the user closes the window */
794    return;
    }
```



```
796  /*****************************************************************
797   * REST_FindHostInArray
798   *
799   * Description:
800   *   This routine will determine if a given host is in an array of
                                                                  hosts.
801   *
802   * Parameters:
803   *   hostName     (I) - name of the host to look for
804   *   sourceHosts (I) - array of hosts to search
805   *
806   * Returns:
807   *   BOOL_TRUE  - If the host is found in the array
808   *   BOOL_FALSE - otherwise
809   *
810   *****************************************************************/
811
812  BoolEnum REST_FindHostInArray (Str    hostName,
813                                 char **sourceHosts)
814  {
815    char      **nextHost;           /* Next host to compare */
816    BoolEnum    found = BOOL_FALSE; /* Flag if we found it */
817
818    /* Loop through all hosts in the array util we find it */
819    nextHost = sourceHosts;
820    while ((*nextHost != NULL) && (!found))
821    {
822
823      /* See if this one matches */
824      if (STR_Cmp (*nextHost, hostName) == CMP_EQUAL)
825        found = BOOL_TRUE;
826      else
827        nextHost++;
828    }
829
830    /* Return whether or not we found it */
831    return (found);
832  }
```

```
834  /*****************************************************************
835   * REST_AddClientInfo
836   *
837   * Description:
838   *    This routine will add a client info object to the current list.
839   *
840   * Parameters:
841   *    info     (I) - the client info to add.
842   *
843   * Returns:
844   *    None.
845   *
846   *****************************************************************/
848  void REST_AddClientInfo (RestoreInfoPtr *currentList,
849                           RestoreInfoPtr  info)
850 1 {
851 1   RestoreInfoPtr tmpInfo;
                                  /* Info to walk the current list with */
852 1   RestoreInfoPtr lastInfo = NULL;    /* Last info we looked at */
853 1   BoolEnum       found = BOOL_FALSE;
                                  /* Flag if we found the insert point */

855 1   /* If this is the first, make it the top */
856 1   if (*currentList == NULL)
857 2   {
858 2     *currentList = info;
859 1   }
860 1   else
861 2   {

863 2     /* Find the insert point */
864 2     tmpInfo = *currentList;
865 2     while ((tmpInfo != NULL) && (!found))
866 3     {

868 3       /* If this is the insert point */
869 3       if (REST_IsLessThan (info, tmpInfo))
870 4       {

872 4         /* Flag that we found the insert point */
873 4         found = BOOL_TRUE;

875 4         /* Insert the new client before the current pointer */
876 4         info->next = tmpInfo;
877 4         if (lastInfo != NULL)
878 4           lastInfo->next = info;
879 4         else
880 4           *currentList = info;
881 4       }
882 3       else
883 4       {
884 4         /* Not this one, save the pointer and go to the next */
885 4         lastInfo = tmpInfo;
886 4         tmpInfo = tmpInfo->next;
887 3       }
888 2     }

890 2   /* If we didn't find an insert point, tack it onto the end */
891 2   if (!found)
892 3   {
893 3     lastInfo->next = info;
894 2   }
895 1 }
896   }
```



```
898  /*****************************************************************
899   * REST_ShowClients
900   *
901   * Description:
902   *    This routine will add clients to the file manager and to the
903   *    current object list.  It will use any clients passed on the
904   *    command line that are indeed backup clients.  If no clients
905   *    are passed on the command line, it will display all restorable
906   *    clients. The list given will be updated to the list of valid
907   *    clients being shown.
908   *
909   *    This routine will then select the workites that are in the
910   *    given work item list.
911   *
912   *    NOTE:
913   *          Currently only one work item may be selected at a time.
914   *          This limitation causes this routine to only select the
915   *          first work item in the given list.
916   *
917   * Parameters:
918   *    clientList (
                    IO) - List of clients to show, updated to valid clients
919   *    wiList     (I)  - List of work items to select
920   *
921   * Returns:
922   *    None.
923   *
924   *****************************************************************/

926  void  REST_ShowClients (RestoreClientPtr   *clientList,
927                          RestoreWorkItemPtr  wiList)
928 1 {
929 1   RestoreInfoPtr    topInfo = NULL;
                                    /* Beginning of client list */
930 1   RestoreClientPtr thisClient;
                                     /* Client info in the list */
931 1   RestoreClientPtr nextClient;
                                   /* Next client info in the list */
932 1   RestoreClientPtr lastClient;
                                   /* Last client info in the list */
933 1   Char             clientName[MAX_CLIENT_NAME_LENGTH];
                                         /* Next client to add */
934 1   RestoreInfoPtr   info;
                                  /* New info object for the client */
935 1   RestoreInfoPtr   tmpInfo;
                                   /* Pointer to walk the list with */
936 1   RestoreInfoPtr   nextInfo;
                                   /* Pointer to next item in list */
937 1   RestoreInfoPtr   lastInfo;
                                   /* Pointer to last item in list */
938 1   long             cookie = INIT_COOKIE; /* Ah, the magic cookie */
939 1   short            numEntries;
                                   /* Number of hosts returned */
940 1   char            *tempHosts[HOSTS_BUFFER_LENGTH];
                                   /* Temporary host array */
941 1   char            **sourceHosts;
                                   /* Array of hosts */
942 1   short            hostCount = 0;
                                   /* Count of restorable hosts */
943 1   Int              index;
                                   /* Loop Counter */
944 1   eertno_ty        eertno;
                                   /* Error status */
945 1   BoolEnum         invalidClient = BOOL_FALSE;
                                   /* Any clients found invalid */
946 1   static BoolEnum  isValidated = BOOL_FALSE;
                                   /* Have we validated the clients */
947 1   BoolEnum         wiFound = BOOL_FALSE;
                                   /* Flag if we found/selected a WI */
```

```
948  1  RestoreWorkItemPtr nextWi;
                                        /* Next WI to find from given list */
949  1  RestoreInfoPtr   wiInfo;
                                        /* The work item that was found */
950  1  RestoreInfoPtr   clientInfo;
                                        /* Next client info in the list */
952  1  if (!isValidated)
953  2  {
955  2    isValidated = BOOL_TRUE;
957  2    /*
958  2     * Get the number of restorable hosts
959  2     */
961  2    /* Allocate space for the temporary hosts */
962  2    for (index=0; index < HOSTS_BUFFER_LENGTH; index++)
963  3      tempHosts[index] = (char *) GUTIL_Malloc (
                                    MAX_CLIENT_NAME_LENGTH * sizeof(char));
965  2    /* Get the source hosts and count them */
966  2    while (cookie != DONE_COOKIE)
967  3    {
968  3      if ((eerrno = EDMRST_GetSourceHosts (GREST_Handle,
969  3                                           REST_RestoreClient,
970  3                                           HOSTS_BUFFER_LENGTH,
971  3                                           tempHosts,
972  3                                           &numEntries,
973  3                                           &cookie)) == 0)
974  4      {
975  4        /* Add this count to the host count */
976  4        hostCount += numEntries;
977  3      }
978  3      else
979  4      {
980  4        /* Got an error, ignore it and get out */
982  4        cookie = DONE_COOKIE;
983  3      }
985  2    }
986  2    /* We're done with the temporary hosts, free 'em up */
987  2    for (index=0; index < HOSTS_BUFFER_LENGTH; index++)
988  3      GUTIL_Free (tempHosts[index]);
990  2    /* Now, really get the restorable hosts */
991  2    if (hostCount > 0)
992  3    {
994  3      /* Allocate enough space for the restorable hosts */
995  3      sourceHosts = (char **) GUTIL_Malloc ((hostCount + 1) * sizeof(
                                                        char *));
996  3      for (index=0; index < hostCount; index++)
997  4        sourceHosts[index] = (char *) GUTIL_Malloc (
                                    MAX_CLIENT_NAME_LENGTH * sizeof(char));
999  3      /* Set the last one to NULL */
1000 3      sourceHosts[hostCount] = NULL;
1002 3      /* Get the hosts (
                              no need to loop, we know how many there are) */
1003 3      cookie = INIT_COOKIE;
1004 3      EDMRST_GetSourceHosts (GREST_Handle,
1005 3                             REST_RestoreClient,
1006 3                             hostCount,
1007 3                             sourceHosts,
1008 3                             &numEntries,
1009 3                             &cookie);
```



```
1010 3    /* Set the last client */
1011 3    lastClient = *clientList;
1013 3    /* If no clients are currently selected, show all */
1014 3    if (*clientList != NULL)
1015 4    {
1016 4      /* Loop through the current hosts,
                                          and add only the backup clients */
1017 4      thisClient = *clientList;
1018 4      while (thisClient != NULL)
1019 5      {
1021 5        /* Save a pointer to the next client */
1022 5        nextClient = thisClient->next;
1024 5        /* Check if this is a backup client */
1025 5  /*
1026 5        if (REST_FindHostInArray (
                                    thisClient->clientName, sourceHosts) &&
1027 5            REST_ValidateClient (thisClient->clientName))
1028 5  */
1029 5        if (REST_FindHostInArray (
                                      thisClient->clientName, sourceHosts))
1030 6        {
1031 6          /* Create the new client info and add it to the list */
1032 6          info = REST_CreateClientInfo(thisClient->clientName);
1033 6          REST_AddClientInfo (&topInfo, info);
1034 6          lastClient = thisClient;
1035 5        }
1036 5        else
1037 6        {
1038 6          if (lastClient != NULL)
1039 6            lastClient->next = thisClient->next;
1040 6          else
1041 6            *clientList = thisClient->next;
1043 6          /* Free this client */
1044 6          GUTIL_Free (thisClient);
1046 6          /* Flag if the client was invalid */
1047 9          if (REST_FindHostInArray (
                                      thisClient->clientName, sourceHosts))
1048 7          {
1049 7            invalidClient = BOOL_TRUE;
1050 6          }
1051 5        }
1053 5        /* Go to the next host */
1054 5        thisClient = nextClient;
1055 4      }
1056 3    }
1058 3    /*
1059 3     * If no clients were selected or none selected are restoreable
1060 3     *   Just show all available hosts (better than none)
1061 3     */
1061 3    if (topInfo == NULL)
1064 4    {
1066 4      /* Loop through all the source hosts */
1067 4      for (index = 0; index < hostCount; index++)
1068 5      {
```

```
/*
            if (REST_ValidateClient (sourceHosts[index]))
*/
if (BOOL_TRUE)
            {
              /* Create the new client info and add it to the list */
              info = REST_CreateClientInfo(sourceHosts[index]);
              REST_AddClientInfo (&topInfo, info);

              /* Create the new client */
              nextClient = (RestoreClientPtr) GUTIL_Malloc (sizeof(
                                                    RestoreClientRec));
              STR_Cpy (nextClient->clientName, sourceHosts[index]);
              nextClient->next = NULL;

              /* Update the client list */
              if (lastClient != NULL)
                lastClient->next = nextClient;
              else
                *clientList = nextClient;

              lastClient = nextClient;
            }
            else
            {
              invalidClient = BOOL_TRUE;
            }
          }
        }

        /* Free up the hosts and the data */
        for (index=0; index < hostCount; index++)
          GUTIL_Free (sourceHosts[index]);
        GUTIL_Free (sourceHosts);
      }
    }
    else
    {
      /* Loop through the current hosts we know they are all valid */
      thisClient = *clientList;
      while (thisClient != NULL)
      {

        /* Create the new client info and add it to the list */
        info = REST_CreateClientInfo(thisClient->clientName);
        REST_AddClientInfo (&topInfo, info);

        /* Go to the next host */
        thisClient = thisClient->next;
      }
    }

  /* If there are still no clients restoreable */
  if (topInfo == NULL)
  {
    /* It there were only invalid clients,
                                          display the no data error */
    if (invalidClient)
    {
      GALERT_DisplayError ((WinPtr)REST_RestoreWin,
                           REST_GetErrorString (REST_ERROR_INDEX),
                           GICON_GetError(),
                           REST_GetErrorString (
                                          REST_NO_RESTORE_DATA_ERROR));
    }
    /* Else, display the permission denied error */
```



```
    else
    {
      GALERT_DisplayError ((WinPtr)REST_RestoreWin,
                           REST_GetErrorString (REST_ERROR_INDEX),
                           GICON_GetError(),
                           REST_GetErrorString (
                                          REST_PERMISSION_ERROR));
    }
    GUTIL_Shutdown ();
  }

  /* Add all the clients to the File Manager */
  tmpInfo = topInfo;
  while (tmpInfo != NULL)
  {
    GFMGR_AddChild (fileMgrContext, NULL, (GFMGR_Object)tmpInfo);
    tmpInfo = tmpInfo->next;
  }

  /* If any workitems were passed in, find and select them */
  if (wiList != NULL)
  {
    /*
     * NOTE:
     *
     * This should loop through all selected work items when we are
     * able to multi-select work items.
     */

    nextWi = wiList;
    clientInfo = topInfo;
    while (!wiFound && (clientInfo != NULL))
    {
      REST_CreateInfoChildren (clientInfo);
      wiInfo = REST_FindInfoInChildren (
                                        nextWi->wiName, clientInfo, BOOL_TRUE);
      if (wiInfo != NULL)
      {
        /* Flag that we found the work item */
        wiFound = BOOL_TRUE;

        /* Show the workitems for this client in the file manager */
        GFMGR_OpenObject (fileMgrContext, (GFMGR_Object)clientInfo);
      }
      else
      {
        /* Go on to the next client */
        clientInfo = clientInfo->next;
      }
    }
  }

  /* Select the first host, unless a work item was selected */
  if (!wiFound)
  {
    GFMGR_SelectObject (fileMgrContext, topInfo, BOOL_TRUE);
    REST_UpdateBackupOptions (NULL);
  }
  else
  {
    /* Select the work item in the file manager */
    GFMGR_SelectObject (fileMgrContext, (
                                        GFMGR_Object)wiInfo, BOOL_TRUE);
    REST_UpdateBackupOptions (wiInfo);
  }
}
```

```
/*****************************************************************
 * REST_VerifySelection
 *
 * Description:
 *    This routine is provided for the file manager.  It determines
 *    if it is OK to select the given object.  We must verify with
 *    the user before switching WIs if anything is marked or
 *    if the search window is displayed (
                                          due to the restore API limitation
 *    of one workitem at a time).
 *
 * Parameters:
 *    selectedObject (I) - Object to verify selection for
 *    isMultiSelect  (I) - Flag is this is a multi select
 *
 * Returns:
 *    BOOL_TRUE  - If it is OK to select the object
 *    BOOL_FALSE - otherwise
 *
 *****************************************************************/

BoolEnum REST_VerifySelection (GFMGR_Context fMgr,
                               GFMGR_Object  selectedObject,
                               BoolEnum      isMultiSelect,
                               BoolEnum      isOpenObject)
{
   RestoreInfoPtr   info;
                                /* Real representation of the object */
   RestoreInfoPtr   tmpObject;
                                        /* Pointer to walk the list with */
   RestoreInfoPtr   newWorkItem = NULL;
                                /* Work item for the selected object */
   BoolEnum         found = BOOL_FALSE; /* Flag if we found it */
   Char             outputString[MAX_STRING_LENGTH];
                                                  /* Error string */
   BoolEnum         removeSearch = BOOL_FALSE;
                                          /* Should search be removed */
   BoolEnum         returnStatus = BOOL_TRUE;  /* Status to return */

   /* If a restore is in progress, no selecting is allowed! */
   if (REST_RestoreInProgress() || REST_SearchInProgress())
     return (BOOL_FALSE);

   /* Get the object in the real data type */
   info = (RestoreInfoPtr)selectedObject;

   /* Go to the top of the selected box,
                                        so we can check if anything is marked */
   LBOX_GoHome (REST_RestoreWin->SelectedListBox);

   /*
    * Quick check,
    *  If the user selected nothing it is always OK to switch.
    *  If the search window is not displayed and nothing is marked it
                                                                  is
    *  also OK to switch.
    */

   if ((info == NULL) ||
       (!REST_SearchWindowIsDisplayed () &&
        LBOX_CurGetClientData(
                                REST_RestoreWin->SelectedListBox) == NULL))
   {
     return (BOOL_TRUE);
   }
```



```
   /*
    * Quick check,
    *  If the user selected a client and it is the current client
    *  it is OK to switch.
    */

   if ((info >type == REST_Client) && (
                                                 info == REST_GetCurrentClientInfo ()))
   {
     return (BOOL_TRUE);
   }

   /* Find the work-item object for this object */
   tmpObject = info;
   while ((newWorkItem == NULL) && (tmpObject != NULL))
   {
     /* If this is the work-item object we're done */
     if (tmpObject->type == REST_WorkItem)
     {
       newWorkItem = tmpObject;
     }

     /* Else try its parent */
     else
     {
       tmpObject = tmpObject->parent;
     }
   }

   /* If the search window is displayed, ask user first */
   if (REST_SearchWindowIsDisplayed () &&
       (newWorkItem != currentWorkItemInfo))
   {
     if (info->type == REST_Client)
     {
       STR_Cpy (outputString,
                REST_GetErrorString (
                                        REST_SWITCH_CLIENT_SEARCH_WARNING));
     }
     else
     {
       STR_Cpy (outputString,
                REST_GetErrorString (REST_SWITCH_WIS_SEARCH_WARNING));
     }

     if (GALERT_DisplayQuestion ((WinPtr)REST_RestoreWin,
                                 REST_GetErrorString (
                                                        REST_WARNING_INDEX),
                                 GICON_GetWarning(),
                                 outputString,
                                 BOOL_FALSE) == GALERT_Affirmative)
     {
       removeSearch = BOOL_TRUE;
     }
     else
     {
       return (BOOL_FALSE);
     }
   }

   /*
    * If something is marked, verify if we have switched WIs
    */

   if (LBOX_CurGetClientData(REST_RestoreWin->SelectedListBox) != NULL)
```

```
1315 2  {
1317 2    /* For client selection, ask user before switching clients */
1318 2    if (info->type == REST_Client)
1319 3    {
1321 3      if (GALERT_DisplayQuestion ((WinPtr)REST_RestoreWin,
1322 3                                  REST_GetErrorString (
                                                REST_WARNING_INDEX),
1323 3                                  GICON_GetWarning(),
1324 3                                  REST_GetErrorString (
                                                REST_SWITCH_CLIENT_WARNING),
1325 3                                  BOOL_FALSE) != GALERT_Affirmative)
1326 4      {
1327 4        returnStatus = BOOL_FALSE;
1328 3      }
1329 2    }
1331 2    /* If we are looking at a work-item already,
                                          see if it is the same one */
1332 2    else if (currentWorkItemInfo != NULL)
1333 3    {
1335 3      /* If this is a different Work-Item, ask before switching */
1336 3      if (newWorkItem != currentWorkItemInfo)
1337 4      {
1338 4        if (GALERT_DisplayQuestion ((WinPtr)REST_RestoreWin,
1339 4                                    REST_GetErrorString (
                                                REST_WARNING_INDEX),
1340 4                                    GICON_GetWarning(),
1341 4                                    REST_GetErrorString (
                                                REST_SWITCH_WIS_WARNING),
1342 4                                    BOOL_FALSE) != GALERT_Affirmative)
1343 5        {
1344 5          returnStatus = BOOL_FALSE;
1345 4        }
1346 3      }
1347 2    }
1348 1  }
1350 1  /* Remove the search dialog if necessary */
1351 1  if (returnStatus && removeSearch)
1352 1    REST_SearchRemove ();
1354 1  /* Return the determined status */
1355 1  return (returnStatus);
1356   }
```



```
1358  /*****************************************************************
1359   * REST_LBoxSelectionNfy
1360   *
1361   * Description:
1362   *   This routine will sensitize and desensize the mark and unmark
                                                              buttons
1363   *   based on the file manager list box selection status.
1364   *
1365   * Parameters:
1366   *   None.
1367   *
1368   * Returns:
1369   *   None.
1370   *
1371   *****************************************************************/
1373  void REST_LBoxSelectionNfy (GFMGR_Context fMgr)
1374  {
1375 1  RestoreInfoPtr   info;
                                        /* Next selected info in the lbox */
1376 1  BoolEnum         mark = BOOL_FALSE;
                                        /* Senitivity of the mark button */
1377 1  BoolEnum         unmark = BOOL_FALSE;
                                        /* Senitivity of the unmark button */
1378 1  boolean_ty       isMarkable = FALSE;
                                        /* Flag if object is markable */
1379 1  Int              numWorkItems = 0;
                                        /* Number of workitems selected */
1381 1  /* If a restore is in progress, don't update any sensitivity */
1382 1  if (REST_RestoreInProgress() || REST_SearchInProgress())
1383 1    return;
1385 1  /* Get all items that are highlighted */
1386 1  info = (RestoreInfoPtr)GFMGR_GetFirstSelectedLBoxObject (
                                                        fileMgrContext);
1387 1  while (info != NULL)
1388 2  {
1390 2    /* If this object is marked, sensitize the unmark button */
1391 2    if (info->marked)
1392 3    {
1393 3      unmark = BOOL_TRUE;
1394 2    }
1396 2    /* Else If this is a restore object, check if it is markable */
1397 2    else if (info->restoreObject != NULL)
1398 3    {
1399 3      if (info->type == REST_WorkItem)
1400 4      {
1401 4        mark = BOOL_TRUE;
1402 4        numWorkItems++;
1403 3      }
1404 3      else if (GREST_IsObjectMarkable (
                                        GREST_Handle, info->restoreObject) &&
1405 3               ((info->status != Backup_Bad) || REST_MarkBadFiles))
1406 4      {
1407 4        /* If this object is markable, sensitize the mark button */
1408 4        mark = BOOL_TRUE;
1409 3      }
1410 2    }
1412 2    /* get the next selected row */
1413 2    info = (RestoreInfoPtr)GFMGR_GetNextSelectedLBoxObject (
                                                        fileMgrContext);
```

```
1414 1    }
1415 1
1416 1    /* Set the sensitivity of the buttons */
1417 1    mark &= (numWorkItems <= 1);
1418 1    WGT_SetEnabled ((WgtPtr)REST_RestoreWin->MarkButton, mark);
1419 1    WGT_SetEnabled ((WgtPtr)REST_RestoreWin->UnmarkButton, unmark);
1420      }
```



```
1422  /*****************************************************************
1423   * REST_FileSelectionNfy
1424   *
1425   * Description:
1426   *   This routine will handle the selection of an object in the file
1427   *   manager scrolled area.
1428   *
1429   * Parameters:
1430   *   selectedObject (I) - the object that was selected.
1431   *
1432   * Returns:
1433   *   None.
1434   *
1435   *****************************************************************/
1437  void REST_FileSelectionNfy (GFMGR_Context fMgr,
1438                              BoolEnum      isOpenObject)
1439 1{
1440 1  RestoreInfoPtr    info;
                                  /* The real data type for the object */
1441 1  errno_ty          eerrno;     /* Error code */
1442 1  RestoreObjectType objectType; /* Object type of the object */
1443 1  Str               fullPath;
                                  /* Copy of the full path of the object */

1445 1  /* If we are sorting or re-reading, do nothing */
1446 1  if (REST_IsReReadInProgress())
1447 1    return;

1449 1  /* Get the real info */
1450 1  info = (RestoreInfoPtr) GFMGR_GetFirstSelectedObject (
                                                        fileMgrContext);

1452 1  /* Verify that somethin is selected */
1453 1  if (info != NULL)
1454 2  {

1456 2    /*
1457 2     * Grab a copy of the full path and type incase we change
                                                        workItems and

1458 2     * invalidate the current objects
1459 2     */

1461 2    fullPath = esl_strdup (REST_GetFullName(info));
1462 2    objectType = info->type;

1464 2    /* Save the name of the object */
1465 2    REST_SetSelectionString (fullPath);

1467 2    /* Update the backup options based on this object */
1468 2    REST_UpdateBackupOptions (info);

1470 2    /* Update buttons based on what is selected in the LBox */
1471 2    REST_LBoxSelectionNfy (NULL);

1473 2    /* Set the current path to the selected item */
1474 2    if ((objectType == REST_Client) || (objectType == REST_WorkItem))
1475 2      REST_ShowPath ("");
1476 2    else
1477 2      REST_ShowPath (fullPath);

1479 2    GUTIL_Free (fullPath);
1480 1  }
1481  }
```

```
1483  /*****************************************************************
1484   * REST_IsMarkable
1485   *
1486   * Description:
1487   *   This routine will determine if the given object is 'markable'.
1488   *
1489   * Parameters:
1490   *   fileObject (I) - the object to check out
1491   *
1492   * Returns:
1493   *   None.
1494   *
1495   *****************************************************************/

1497  static BoolEnum REST_IsMarkable( GPMGR_Context fMgr,
1498                                   GPMGR_Object  fileObject )
1499 1{
1500 1  RestoreInfoPtr info;      /* The real info for the object */
1501 1  BoolEnum       retVal;    /* Value to return */


1503 1  /* Cast back to the real object */
1504 1  info = (RestoreInfoPtr)fileObject;


1506 1  /* Clients workitems and errors are not markable */
1507 1  if ((info->type == REST_Client) ||
1508 1      (info->type == REST_FailedWorkItem) ||
1509 1      (info->type == REST_ErrorObject))
1510 2  {
1511 2    retVal = BOOL_FALSE;
1512 1  }
1513 1  else
1514 2  {
1515 2    retVal = BOOL_TRUE;
1516 1  }

1518 1  return (retVal);
1519  }
```



```
1521  /*****************************************************************
1522   * REST_IsMarked
1523   *
1524   * Description:
1525   *   This routine will determine if the given object is 'marked'.
1526   *
1527   * Parameters:
1528   *   fileObject (I) - the object to check out
1529   *
1530   * Returns:
1531   *   None.
1532   *
1533   *****************************************************************/

1535  static BoolEnum REST_IsMarked( GPMGR_Context fMgr,
1536                                 GPMGR_Object  fileObject )
1537 1{
1538 1  RestoreInfoPtr info;      /* The real info for the object */


1540 1  /* Cast back to the real object */
1541 1  info = (RestoreInfoPtr)fileObject;

1543 1  return (info->marked);
1544  }
```

```
1546  /*****************************************************************
1547   * REST_HasMarkedChildren
1548   *
1549   * Description:
1550   *    This routine will determine if the given object has any marked
                                                              children.
1551   *    This only returns true if the children exist,
                                                  it does not retrieve new
1552   *    children.
1553   *
1554   * Parameters:
1555   *    parent (I) - the parent to check
1556   *
1557   * Returns:
1558   *    None.
1559   *
1560   *****************************************************************/
1562  static BoolEnum REST_HasMarkedChildren (RestoreInfoPtr parent)
1563 1{
1564 1  BoolEnum        retVal = BOOL_FALSE;
1565 1  RestoreInfoPtr nextChild;

1567 1  nextChild = parent->children;
1568 1  while (!retVal && (nextChild != NULL))
1569 2  {
1570 2    if (nextChild->marked)
1571 3    {
1572 3      retVal = BOOL_TRUE;
1573 2    }
1574 2    else
1575 3    {
1576 3      retVal = REST_HasMarkedChildren (nextChild);
1577 3      nextChild = nextChild->next;
1578 2    }
1579 1  }

1581 1  return (retVal);
1582  }
```



```
1584  /*****************************************************************
1585   * REST_IsPartialMarked
1586   *
1587   * Description:
1588   *    This routine will determine if the given object is 'marked'.
1589   *
1590   * Parameters:
1591   *    fileObject (I) - the object to check out
1592   *
1593   * Returns:
1594   *    None.
1595   *
1596   *****************************************************************/

1598  static BoolEnum REST_IsPartialMarked( CFMGR_Context fMgr,
1599                                        CFMGR_Object  fileObject )
1600 1{
1601 1  RestoreInfoPtr info;       /* The real info for the object */
1602 1  BoolEnum       retVal = BOOL_FALSE;
1603 1  RestoreInfoPtr nextInfo;
1604 1  RestoreInfoPtr childInfo;
1605 1  Str            fullName;

1607 1  /* Cast back to the real object */
1608 1  info = (RestoreInfoPtr)fileObject;

1610 1  if (info->restoreObject != NULL)
1611 2  {
1612 2    /* Determine if any existing children are marked */
1613 2    retVal = REST_HasMarkedChildren (info);

1615 2    if (retVal != BOOL_TRUE)
1616 3    {
1617 3      /* If there are no marks, then nothing is partially marked */
1618 3      LBOX_GoHome (REST_RestoreWin->SelectedListBox);
1619 3      nextInfo = LBOX_CurGetClientData(
                                          REST_RestoreWin->SelectedListBox);

1620 3      while (!retVal && (nextInfo != NULL))
1621 4      {
1622 4        fullName = esl_strdup (REST_GetFullName(nextInfo));
1623 4        childInfo = REST_FindInfoInChildren (
                                        fullName, info, BOOL_FALSE);

1624 4        if ((childInfo != NULL) && (childInfo->marked))
1625 5        {
1626 5          retVal = BOOL_TRUE;
1627 4        }
1628 4        else
1629 5        {
1630 5          LBOX_GoDown (REST_RestoreWin->SelectedListBox);
1631 5          nextInfo = LBOX_CurGetClientData(
                                          REST_RestoreWin->SelectedListBox);
1632 4        }
1633 4        GUTIL_Free (fullName);
1634 3      }
1635 2    }
1636 1  }

1638 1  return (retVal);
1639  }
```

```
1641  /*****************************************************************
1642   * REST_ToggleMarkNfy
1643   *
1644   * Description:
1645   *    This routine will toggle the mark status of the given object.
1646   *
1647   * Parameters:
1648   *    fileObject (I) - The file manager object to toggle
1649   *
1650   * Returns:
1651   *    None.
1652   *
1653   *****************************************************************/

1655  void REST_ToggleMarkNfy ( GFMGR_Context fMgr,
1656                            GFMGR_Object  fileObject )
1657  1 {
1658  2   RestoreInfoPtr   info;
                                        /* The real info for the object */
1659  1   long             numberBad = 0;
                                        /* Number of bad files marked */
1660  1   long             numberMarked = 0;   /* Number of objects marked */
1661  1   RestoreInfoPtr   tmpObject;
                                        /* Pointer to walk the list with */
1662  1   RestoreInfoPtr   newWorkItem = NULL;
                                        /* Work item for the selected object */
1663  1   RestoreInfoPtr   topInfo;
                                        /* The parent object in the SArea */

1665  1   /* If a restore is in progress, no marking is allowed! */
1666  1   if (REST_RestoreInProgress() || REST_SearchInProgress())
1667  1     return;

1669  1   /* Cast back to the real object */
1670  1   info = (RestoreInfoPtr)fileObject;

1672  1   if ((info != NULL) &&
1673  1       ((currentWorkItemInfo != NULL) || (
                                                info->type == REST_WorkItem)))
1674  2   {
1675  2     /* Find the work-item object for this object */
1676  2     tmpObject = info;
1677  2     while ((newWorkItem == NULL) && (tmpObject != NULL))
1678  3     {
1679  3       /* if this is the work-item object we're done */
1680  3       if (tmpObject->type == REST_WorkItem)
1681  4       {
1682  4         newWorkItem = tmpObject;
1683  3       }
1684  3       /* Else try its parent */
1685  3       else
1686  4       {
1687  4         tmpObject = tmpObject->parent;
1688  3       }
1689  2     }

1691  2     /* If this is a different workItem */
1692  2     if (newWorkItem != currentWorkItemInfo)
1693  3     {
1694  3       /* If the client is selected, then we might be able to mark */
1695  3       topInfo = (RestoreInfoPtr) GFMGR_GetFirstSelectedObject (
                                                        fileMgrContext);
1696  3       if ((topInfo->type == REST_Client) &&
1697  3           (info->type == REST_WorkItem) &&
1698  3           (REST_VerifySelection (fileMgrContext,
```



```
1699  3                                  newWorkItem,
1700  3                                  BOOL_FALSE,
1701  3                                  BOOL_FALSE)))
1702  4       {
1703  4         /*
1704  4          * Select the work item,
                                              this causes it to be selected in the SArea
1705  4          * which isn't really the desired result,
                                                    but we have to have the
1706  4          * workitem in the SArea in order to continue
1707  4          */

1709  4         GFMGR_SelectObject (fileMgrContext, (
                                                    GFMGR_Object)info, BOOL_TRUE);
1710  3       }
1711  3       else
1712  4       {
1713  4         return;
1714  3       }
1715  2     }

1717  2     /* This may take a while so display a wait cursor */
1718  2     WGT_UseWaitCursor ((WgtPtr)REST_RestoreWin);

1720  2     /*
1721  3      * Continue with the operation if:
1722  3      *  The object is marked (anything can be unmarked)
1723  3      *  The object is a higher level object (like a workitem)
1724  3      *  The object is markable for the given time and passes the bad
                                                              file test
1725  2      */

1727  2     if ((info->marked) ||
1728  2         ((info->type != REST_File) && (
                                                  info->type != REST_Directory)) ||
1729  2         (GREST_IsObjectMarkable (GREST_Handle, info->restoreObject) &&
1730  2          (info->status != Backup_Bad) || REST_MarkBadFiles)))
1731  3     {
1732  3       if ((info->marked && REST_UnmarkInfo (
                                                  info, &numberMarked, &numberBad)) ||
1733  3           (!info->marked && REST_MarkInfo (
                                                  info, &numberMarked, &numberBad)))
1734  4       {
1735  4         REST_UpdateMarkedDataFromList (numberMarked, numberBad);
1736  4         SAREA_RedrawParts ((SAreaPtr)REST_RestoreWin->SelectedListBox);
1737  4         REST_UpdateRemoveButtons ();
1738  3       }
1739  2     }

1741  2     /* Display the normal cursor again */
1742  2     WGT_UseCursor ((WgtPtr)REST_RestoreWin, CURS_Arrow());
1743  1   }
1744  }
```

```
1746  /*****************************************************************
1747   * REST_FileInitialize
1748   *
1749   * Description:
1750   *    This routine will initialize the routines in the file manager
                                                                    portion
1751   *    of the restore functionality.
1752   *
1753   * Parameters:
1754   *    None.
1755   *
1756   * Returns:
1757   *    None.
1758   *
1759   *****************************************************************/

1761  void REST_FileInitialize (void)
1762  {

1764    /* Create the file manager context for restore */
1765    fileMgrContext = GFMGR_Create
1766                        (REST_RestoreWin->BackupSArea,
1767                         REST_RestoreWin->BackupListBox,
1768                         REST_RestoreWin->LBoxVSb,
1769                         (TEdPtr)REST_RestoreWin->JunkTEd,
1770                         NUMBER_ITEMS_COLUMNS,
1771                         BOOL_FALSE,
1772                         REST_GetChildren,
1773                         REST_CloseChildren,
1774                         REST_GetIcon,
1775                         REST_GetOverlayIcon,
1776                         REST_GetIcon2,
1777                         REST_GetOverlayIcon2,
1778                         REST_GetNameString,
1779                         REST_HasChildren,
1780                         REST_HasChildren,
1781                         REST_GetBackupCellString,
1782                         REST_VerifySelection,
1783                         REST_FileSelectionNfy,
1784                         REST_LBoxSelectionNfy,
1785                         REST_IsMarkable,
1786                         REST_IsMarked,
1787                         REST_ToggleMarkNfy,
1788                         REST_IsPartialMarked);

1790  }
```

```
/*****************************************************************
 * restProgressMgr.c
 *
 *
 * Copyright 1999 by EMC Corp.
 *
 *
 * Mission Statement:
 *   This file contains the functions necessary for the EDM Restore
 *   Progress window.
 *
 *
 * Required includes:
 *   None
 *
 * Compile-Time Options:
 *   N/A
 *
 *
 * RCS Information:
 *      $RCSfile$
 *      $Revision$
 *      $Date$
 *****************************************************************/

#define  ERR_LIB            RESTORE

#include <esl/c_portable.h>
#include <esl/ep_xOpen.h>

#include <respub.h>
#include <cboxpub.h>
#include <panelpub.h>
#include <tbutpub.h>
#include <winpub.h>
#include <drawpub.h>

#include "eerrno/e_errno.h"
#include "util/esl_string.h"
#include <restore/restore_api.h>
#include <restore/restore_engine.h>

#include "gutil/timeUtils.h"
#include "gutil/alertMgr.h"
#include "gutil/iconDefs.h"
#include "gutil/iconUtils.h"
#include "gutil/winUtils.h"
#include "gutil/lboxUtils.h"
#include "gutil/guiUtils.h"
#include "gutil/panel.h"

#include "restoreP.h"
#include "restUtils.h"
#include "restCBMgr.h"
#include "restSearch.h"
#include "restFileMgr.h"
#include "restQuestion.h"
#include "restProgress.h"

/*************
 * Constants *
 *************/

#define KBYTES  (1 << 10)
#define MBYTES  (1 << 20)
```



```
/* All delays are in milli-seconds */
#define RESTORE_DONE_DELAY          1
#define RESTORE_CHECK_CANCEL_DELAY 5000
#define RESTORE_CHECK_SUBMIT_DELAY  500
#define RESTORE_CHECK_STARTUP_DELAY 500

#define HORIZONTAL_MARGIN  5
#define VERTICAL_MARGIN    2

#define SEPARATOR_CHAR  '='

/************************
 * Local Data structures *
 ************************/

/********************
 * Global Variables *
 ********************/

static BoolEnum REST_ProgressDisplayed = BOOL_FALSE;
static RestProgressWinPtr REST_ProgressWindow = NULL;

/* Flags for restoral progress */
static BoolEnum restoreInProgress = BOOL_FALSE;
static BoolEnum restoreCancelled = BOOL_FALSE;

#define NUM_WAIT_ICONS      5
#define UPDATE_ICON_COUNT   1
static IconPtr  waitIcon[NUM_WAIT_ICONS];
static Int      iconNum = 0;
static Int      iconCount = 0;

static Int      checkForCancelId = 0;

static time_t  REST_FirstDataTime = 0;

/*****************************************************************
 * REST_SetRestoreVisibility
 *
 * Description:
 *   This routine will set the visibility status of widgets based on
 *   whether or not a restore is in progress.
 *
 * Parameters:
 *   visible (I) - Value of the visibility to set
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void  REST_SetRestoreVisibility (BoolEnum visible)
{
  BoolEnum origStatus; /* Original status of reread flag */

  /* If the state is not visible,
                              disable options not valid during restore */
  if (!visible)
  {
    /* Restrict all actions in the FS Options Panel */
    GUTIL_WGT_SetEnabled ((
                    WgtPtr)REST_RestoreWin->FSOptionsPanel, BOOL_FALSE);

    /* Restrict marking, unmarking, and searching */
    GUTIL_WGT_SetEnabled ((
                    WgtPtr)REST_RestoreWin->MarkButton, BOOL_FALSE);
```

```
110 2     GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_RestoreWin->UnmarkButton, BOOL_FALSE);
111 2     GUTIL_WGT_SetEnabled ((
                      WgtPtr)REST_RestoreWin->SearchButton, BOOL_FALSE);

113 2     /* Restrict unmarking from the mark summary */
114 2     GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_RestoreWin->RemoveButton, BOOL_FALSE);
115 2     GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_RestoreWin->ClearButton, BOOL_FALSE);

117 2     /* Don't let another restore start */
118 2     GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_RestoreWin->StartButton, BOOL_FALSE);

120 2     /* Don't let the user close the window */
121 2     GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_RestoreWin->CloseButton, BOOL_FALSE);


123 2     /* Don't let the user change the path */
124 2     GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_RestoreWin->PathTEd, BOOL_FALSE);


126 2     /* Don't let the user do anything in the search window */
127 2     if (REST_SearchWindowIsDisplayed ())
128 3     {
129 3       GUTIL_WGT_SetEnabled((
                      WgtPtr)REST_SearchWindow->MarkButton,   BOOL_FALSE);
130 3       GUTIL_WGT_SetEnabled((
                      WgtPtr)REST_SearchWindow->UnMarkButton, BOOL_FALSE);
131 3       GUTIL_WGT_SetEnabled((
                      WgtPtr)REST_SearchWindow->SetViewButton,BOOL_FALSE);
132 2     }

134 1   }

136 1   /* Else, set the states back to what they should be */
137 1   else
138 2   {
139 2     /* If we are doing FS restore,
                                    re-enable FS panel and search button */
140 2     if (currentWorkItemInfo != NULL)
141 3     {
142 3       GUTIL_WGT_SetEnabled ((
                      WgtPtr)REST_RestoreWin->FSOptionsPanel, BOOL_TRUE);
143 3       GUTIL_WGT_SetEnabled ((
                      WgtPtr)REST_RestoreWin->SearchButton, BOOL_TRUE);

145 3       /*
146 3        * Sensitive the date buttons appropriately
147 3        * Temporarily make sure the reread flag is set so no error is
                                                              displayed
148 3        */

150 3       origStatus = REST_IsReReadInProgress ();
151 3       REST_SetReReadInProgress (BOOL_TRUE);
152 3       REST_UpdateDateButtons ();
153 3       REST_SetReReadInProgress (origStatus);
154 2     }

156 2     /* Reset the enabling of the mark an unmark buttons */
157 2     REST_LBoxSelectionNfy (NULL);

159 2     /* Reset unmarking from the mark summary */
160 2     REST_UpdateRemoveButtons ();
```



```
162 2     /* Reset the start button */
163 2     GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_RestoreWin->StartButton, BOOL_TRUE);

185 2     /* Reset the Close button */
186 2     GUTIL_WGT_SetEnabled ((
                        WgtPtr)REST_RestoreWin->CloseButton, BOOL_TRUE);

188 2     /* Reset the Path Ted */
189 2     GUTIL_WGT_SetEnabled ((WgtPtr)REST_RestoreWin->PathTEd, BOOL_TRUE);

191 2     /* Reset the search window buttons */
192 2     if (REST_SearchWindowIsDisplayed ())
193 3     {
194 3       REST_SearchUpdateButtons ();
195 2     }
196 1   }
197   }
```

```
199 1  /*****************************************************************
200 1   * REST_ProgressDrawPercentComplete
201 1   *
202 1   * Description:
203 1   *    This routine will draw a graph of % complete in the given drawing
204 1   *    area.
205 1   *
206 1   * Parameters:
207 1   *    drawArea    (I) - The widget to draw to
208 1   *
209 1   * Returns:
210 1   *    None.
211 1   *
212 1   *****************************************************************/

214 1  void  REST_ProgressDrawPercentComplete (WgtPtr drawArea)
215 1  {
216 1    int         percentage;
217 1    Rect16Ptr   wgtBox;
218 1    Rect16Rec   drawBox;
219 1    Rect16Rec   fillBox;
220 1    Char        drawString[256];
221 1    Point16Rec textOri;
222 1    Point16Rec textExt;
223 1    Point16Rec wgtOri;
224 1    Point16Rec wgtExt;
225 1    Point16Rec penExt;
226 1    Rect16Rec   fgBox;
227 1    Rect16Rec   bgBox;
228 1    ColorPtr    fgColor;
229 1    ColorPtr    bgColor;
230 1    errno_ty    status;
231 1    Point16Rec iconExt;
232 1    Rect16Rec   iconRect;

234 1    if (drawArea == NULL)
235 1      return;

237 1    percentage = (int)WGT_GetClientRes (drawArea);

239 1    /*
240 1     * We never want to show 100% complete. Let it finish
241 1     */

243 1    if (percentage >= 100)
244 2    {
245 2      percentage = 99;
246 1    }

248 1    /* Set the clip rectangle to the widget */
249 1    if (DRAW_ClipWgt (drawArea))
250 2    {

252 2      /* Get the size of the pen,
                                    we only want to draw inside the widget's pen */
253 2      PEN_QuerySize (WGT_GetPen (drawArea), &penExt);

255 2      /* Get the box for the entire widget */
256 2      wgtBox = (Rect16Ptr) WGT_GetBox (drawArea);

258 2      /* Get the origin and extents to draw into (
                                                  not overdrawing the pen) */
259 2      wgtOri.x = penExt.x;
260 2      wgtOri.y = penExt.y;
261 2      wgtExt.x = wgtBox->Ext.x - (2 * penExt.x);
```



```
263 2      wgtExt.y = wgtBox->Ext.y - (2 * penExt.y);

264 2      /* Define the rectangle that borders the widget */
265 2      drawBox.Ori.x = wgtOri.x;
266 2      drawBox.Ori.y = wgtOri.y;
267 2      drawBox.Ext.x = wgtExt.x;
268 2      drawBox.Ext.y = wgtExt.y;

270 2      if (restoreInProgress && !restoreCancelled)
271 3      {
272 3        fgColor = COLOR_Transparent ();
273 3        bgColor = WGT_GetBgColor ((WgtPtr)drawArea);
274 2      }
275 2      else
276 3      {
277 3        status = (errno_ty)WGT_GetClientRes (drawArea);
278 3        if (restoreCancelled)
279 4        {
280 4          fgColor = WGT_GetFgColor ((WgtPtr)drawArea);
281 4          bgColor = WGT_GetBgColor ((WgtPtr)drawArea);
282 3        }
283 3        else if (status == E_SUCCESS)
284 4        {
285 4          fgColor = COLOR_Black ();
286 4          bgColor = COLOR_Green ();
287 3        }
288 3        else
289 4        {
290 4          fgColor = COLOR_Black ();
291 4          bgColor = COLOR_Red ();
292 3        }
293 2      }

295 2      /* First draw a box around the entire widget */
296 2      DRAW_SetContext (drawArea,
297 2                       COLOR_Transparent (),
298 2                       bgColor,
299 2                       PEN_Solid(),
300 2                       PATT_Empty(),
301 2                       WGT_GetFont (drawArea));
302 2      DRAW_Rect (drawArea, &drawBox);

304 2      if (restoreInProgress && !restoreCancelled)
305 3      {
306 3        if (percentage >= 0)
307 4        {
308 4          /* Now determine how much to fill in */
309 4          fillBox.Ori.x = wgtOri.x;
310 4          fillBox.Ori.y = wgtOri.y;
311 4          fillBox.Ext.x = (Int16)((float)wgtExt.x * ((
                                                    float)percentage/100.0));
312 4          fillBox.Ext.y = wgtExt.y;

314 4          /* Now draw the filled area depicting the percent complete */
315 4          DRAW_SetContext (drawArea,
316 4                           WGT_GetFgColor (drawArea),
317 4                           WGT_GetFgColor (drawArea),
318 4                           PEN_Solid(),
319 4                           PATT_Empty(),
320 4                           WGT_GetFont (drawArea));
321 4          DRAW_Rect (drawArea, &fillBox);
322 3        }

324 3        /* Determine the string to draw and its size */
325 3        if (percentage >= 0)
326 4        {
```

```
327 4           STR_Sprintf (drawString,
328 4                        REST_GetErrorString (REST_PERCENT_PROGRESS),
329 4                        percentage);
330 3         }
331 3         else
332 4         {
333 5           STR_Sprintf (drawString, REST_GetErrorString (
                                                REST_STARTUP_IN_PROGRESS));
334 3         }

336 3         DRAW_QueryTextExt (drawArea, drawString, &textExt);

338 3         /* Set the origin of the text so that is it displayed in the
                                                                  center */
339 3         textOri.x = (wgtExt.x - textExt.x) / 2;
340 3         textOri.y = penExt.y + ((wgtExt.y - textExt.y) / 2);

342 3         /* First draw the string in the foreground color */
343 3         fgBox.Ori.x = textOri.x;
344 3         fgBox.Ori.y = textOri.y;
345 3         fgBox.Ext.x = textExt.x;
346 3         fgBox.Ext.y = textExt.y;

348 3         DRAW_SetContext (drawArea,
349 3                          WGT_GetFgColor (drawArea),
350 3                          WGT_GetBgColor (drawArea),
351 3                          PEN_Solid(),
352 3                          PATT_Empty(),
353 3                          WGT_GetFont (drawArea));
354 3         DRAW_ClippedText (drawArea,
355 3                           &fgBox,
356 3                           DRAW_JUSTLEFT | DRAW_JUSTVCENTER,
357 3                           drawString)

359 3         /*
360 3          * If the filled area overlaps the text area, we need to draw
361 3          * the portion that overlaps in the background color
362 3          */

364 3         if ((percentage >= 0) && (textOri.x < fillBox.Ext.x))
365 4         {
366 4           /* Set the background box to be the entire string */
367 4           bgBox.Ori.x = textOri.x;
368 4           bgBox.Ori.y = textOri.y;
369 4           bgBox.Ext.x = textExt.x;
370 4           bgBox.Ext.y = textExt.y;

372 4           /*
373 4            * If the filled area ends before the end of the string,
                                                              change the
374 4            * box to end at the end of the filled area
375 4            */

377 4           if (bgBox.Ext.x > ((
                                  fillBox.Ori.x + fillBox.Ext.x) - textOri.x))
378 4             bgBox.Ext.x = ((fillBox.Ori.x + fillBox.Ext.x) - textOri.x;

380 4           DRAW_SetContext (drawArea,
381 4                            WGT_GetBgColor (drawArea),
382 4                            WGT_GetFgColor (drawArea),
383 4                            PEN_Solid(),
384 4                            PATT_Empty(),
385 4                            WGT_GetFont (drawArea));
386 4           DRAW_ClippedText (drawArea,
387 4                             &bgBox,
388 4                             DRAW_JUSTLEFT | DRAW_JUSTVCENTER,
```



```
389 4                             drawString)
390 3         }
391 3         else if (percentage < 0)
392 4         {
393 4           /* draw the icon */
394 4           ICON_QuerySize (waitIcon[iconNum], &iconExt);

396 4           /* This defines the box in which we can place the icons so far
                                                                        */
397 4           iconRect.Ori.x = fgBox.Ori.x + fgBox.Ext.x + HORIZONTAL_MARGIN;
398 4           iconRect.Ori.y = (wgtExt.y - iconExt.y) / 2;
399 4           iconRect.Ext.x = iconExt.x;
400 4           iconRect.Ext.y = iconExt.y;

402 4           DRAW_Icon (drawArea, &iconRect, waitIcon[iconNum]);

404 4           iconCount++;
405 4           if (iconCount == UPDATE_ICON_COUNT)
406 5           {
407 5             iconCount = 0;
408 5             iconNum++;
409 5             if (iconNum == NUM_WAIT_ICONS)
410 6             {
411 6               iconNum = 0;
412 5             }
413 4           }
414 3         }
415 2       }
416 2       else
417 3       {
418 3         status = (eerrno_t)WGT_GetClientRes (drawArea);

420 3         if (restoreCancelled)
421 4         {
422 4           STR_Cpy (drawString, REST_GetErrorString (REST_CANCEL_TITLE));
423 3         }
424 3         else if (status == E_SUCCESS)
425 4         {
426 4           STR_Cpy (drawString, REST_GetErrorString (
                                                REST_RESTORE_SUCCESS));
427 3         }
428 3         else
429 4         {
430 4           STR_Cpy (drawString, REST_GetErrorString (
                                                REST_RESTORE_FAILURE));
431 3         }

433 3         DRAW_QueryTextExt (drawArea, drawString, &textExt);

435 3         /* Set the origin of the text so that is it displayed in the
                                                                  center */
436 3         textOri.x = (wgtExt.x - textExt.x) / 2;
437 3         textOri.y = penExt.y + ((wgtExt.y   textExt.y) / 2);

439 3         /* First draw the string in the foreground color */
440 3         fgBox.Ori.x = textOri.x;
441 3         fgBox.Ori.y = textOri.y;
442 3         fgBox.Ext.x = textExt.x;
443 3         fgBox.Ext.y = textExt.y;

445 3         DRAW_SetContext (drawArea,
446 3                          fgColor,
447 3                          bgColor,
448 3                          PEN_Solid(),
449 3                          PATT_Empty(),
450 3                          WGT_GetFont (drawArea));
```

```
451 3          DRAW_ClippedText (drawArea,
452 3                            &fgBox,
453 3                            DRAW_JUSTLEFT | DRAW_JUSTVCENTER,
454 3                            drawString)

456 2       }

458 2       /* End clipping */
459 2       DRAW_ClipEnd (drawArea);

461 1    }
462    }
```



```
464    /*****************************************************************
465     * Function Name:   REST_GetDataSizeString()
466     *
467     * Description:
468     *   This function will set the give string to represent the value
469     *   of the number of kbytes given in sizeInKB.
470     *
471     *   If the value is less than 1000,        it will display in KB
472     *   If the value is greater than 1000,     it will display in MB
473     *   If the value is greater than 1000000, it will display in GB
474     *
475     * Parameters:
476     *   (I) dataSize - Size of the data in KB
477     *   (O) sizeString - PREALLOCATED string to put size value into
478     *
479     * Success Outputs and Side Effects:
480     *   None.
481     *
482     *****************************************************************/
483    void REST_GetDataSizeString (u_long sizeInKB,
484                                 Str     sizeString)
485 1  {

487 1    if (sizeInKB < KBYTES)
488 2    {
489 2      /* Display K Bytes */
490 2      STR_Sprintf (sizeString, "%lu KB", sizeInKB);
491 1    }
492 1    else if (sizeInKB < MBYTES)
493 2    {
494 2      /* Display M Bytes with 2 decimal places */
495 2      STR_Sprintf (sizeString, "%.2f MB",
496 2                   (float)sizeInKB / (float)KBYTES);
497 1    }
498 1    else
499 2    {
500 2      /* Display G Bytes with 2 decimal places */
501 2      STR_Sprintf (sizeString, "%.2f GB",
502 2                   (float)sizeInKB / (float)MBYTES);
503 1    }
504    }
```

```
/*****************************************************************
 * REST_ProgressUpdate
 *
 * Description:
 *   This routine will display the Progress window.
 *
 * Parameters:
 *   currentProgress - The progress information to display
 *
 * Returns:
 *   BOOL_TRUE  - if the user hit the cancel button
 *   BOOL_FALSE - otherwise.
 *
 *****************************************************************/

static void  REST_ProgressUpdate (feedbackObjectPtr currentProgress)
{
  Int32             kbSoFar = 0;
  Int32             expectedKB = 0;
  Int32             filesSoFar = 0;
  Int32             expectedFiles = 0;
  EDMProgressPtr    objectPtr;
  Char              sizeDone[16];
  Char              sizeTotal[16];
  Char              outString[64];
  time_t            startTime;
  time_t            currentTime;
  time_t            timeRunning;
  time_t            eta;
  float             percentage;

  /* If the progress window is not displayed, we're done */
  if (!REST_ProgressDisplayed)
    return;

  if (currentProgress != NULL)
  {
    /*
     * Tally up the totals
     */

    objectPtr = EDMRST_GetFirstEDMObject (
                                        GREST_Handle, currentProgress);

    while (objectPtr != NULL)
    {
      kbSoFar += EDMRST_GetObjectEDMTotalKbytesSofar (
                                        GREST_Handle, objectPtr);

      expectedKB -= EDMRST_GetObjectEDMTotalKBExpected (
                                        GREST_Handle, objectPtr);

      filesSoFar += EDMRST_GetObjectEDMTotalFiles (
                                        GREST_Handle, objectPtr);

      expectedFiles+= EDMRST_GetObjectEDMTotalFilesExpected (
                                        GREST_Handle,
                                        objectPtr);

      startTime = EDMRST_GetObjectEDMTimeStarted (
                                        GREST_Handle, objectPtr);

      currentTime = EDMRST_GetObjectEDMCurrTime (
                                        GREST_Handle, objectPtr);

      objectPtr = EDMRST_GetNextEDMObject (GREST_Handle, objectPtr);
    }

    TED_SetStr ((TEdPtr)REST_ProgressWindow->WiNameTEd,
                currentWorkItemInfo->name);

    REST_GetDataSizeString (kbSoFar, sizeDone);
    REST_GetDataSizeString (expectedKB, sizeTotal);

    TED_SetStrf ((TEdPtr)REST_ProgressWindow->StatusTEd,
                 REST_GetErrorString (REST_DATA_SIZE_PROGRESS),
                 sizeDone,
                 sizeTotal);
    TED_InsertStr ((TEdPtr)REST_ProgressWindow->StatusTEd, "\n\n");
    TED_InsertStrf ((TEdPtr)REST_ProgressWindow->StatusTEd,
                    REST_GetErrorString (REST_DATA_FILE_PROGRESS),
                    filesSoFar,
                    expectedFiles);

    if ((kbSoFar > 0) || (filesSoFar > 0))
    {
      /* See if this is the first tick */
      if (REST_FirstDataTime == 0)
        REST_FirstDataTime = currentTime - 1;

      timeRunning = currentTime - REST_FirstDataTime;

      if (expectedKB > 0)
      {
        percentage = (float)kbSoFar / (float)expectedKB;
        if (percentage == 0)
          percentage = (float)1 / (float)1000;

      }
      else
      {
        percentage = (float)1 / (float)1000;
      }

      WGT_SetClientRes ((WgtPtr)REST_ProgressWindow->ProgressArea,
                        (ClientPtr)((int)((100 * percentage) + 0.5)));

      eta = REST_FirstDataTime + (int)((
                                  float)timeRunning / percentage);

      strftime (outString,
                MEDIUM_STRING_LENGTH,
                "%H:%M",
                localtime (&startTime));
      TED_SetStr ((
                   TEdPtr)REST_ProgressWindow->StartTimeTEd, outString);

      strftime (outString,
                MEDIUM_STRING_LENGTH,
                "%H:%M",
                localtime (&eta));
      TED_SetStr ((TEdPtr)REST_ProgressWindow->EndTimeTEd, outString);
    }
  }

  if ((currentProgress == NULL) || ((kbSoFar == 0) && (
                                              filesSoFar == 0)))
  {
    TED_SetStr ((TEdPtr)REST_ProgressWindow->StartTimeTEd,      "");
    TED_SetStr ((TEdPtr)REST_ProgressWindow->EndTimeTEd,        "");
    WGT_SetClientRes ((WgtPtr)REST_ProgressWindow->ProgressArea, (
                                                  ClientPtr)-1);
    REST_FirstDataTime = 0;
  }
}
```

```
625 1   REST_ProgressDrawPercentComplete ((
                                    WgtPtr)REST_ProgressWindow->ProgressArea);
626     }
```



```
628     /*****************************************************************
629      * REST_ProgressDisplay
630      *
631      * Description:
632      *   This routine will display the Progress window.
633      *
634      * Parameters:
635      *   None
636      *
637      * Returns:
638      *   None.
639      *
640      *****************************************************************/

642     void REST_ProgressDisplay ( void )
643   1 {
644   1   if (REST_ProgressDisplayed)
645   2   {
646   2     REST_ProgressRemove ();
647   1   }

649   1   /* Load up the window's resources */
650   1   REST_ProgressWindow = REST_ProgressWindowInit ((
                                                WinPtr)REST_RestoreWin);


652   1   /* Flag the progress window as displayed */
653   1   REST_ProgressDisplayed = BOOL_TRUE;

655   1   /* Add the host to the window title */
656   1   GUTIL_AddHostToWindowTitle ((WinPtr)REST_ProgressWindow);

658   1   TED_SetStr ((TEdPtr)REST_ProgressWindow->WinNameTEd,
659   1               currentWorkItemInfo->name);

661   1   WGT_SetClientRes ((WgtPtr)REST_ProgressWindow->ProgressArea, (
                                                        ClientPtr)-1);

663   1   waitIcon[0] = (IconPtr) RES_Load ("Icons", "Wait1Icon");
664   1   waitIcon[1] = (IconPtr) RES_Load ("Icons", "Wait2Icon");
665   1   waitIcon[2] = (IconPtr) RES_Load ("Icons", "Wait3Icon");
666   1   waitIcon[3] = (IconPtr) RES_Load ("Icons", "Wait4Icon");
667   1   waitIcon[4] = (IconPtr) RES_Load ("Icons", "Wait5Icon");
668   1   iconNum = 0;
669   1   iconCount = 0;

671   1   /* Put up the window */
672   1   WIN_Show ((WinPtr)REST_ProgressWindow);
673     }
```

```
675  /*****************************************************************
676   * REST_ProgressDisplayError
677   *
678   * Description:
679   *   This routine will display an error in the  Progress window.
680   *
681   * Parameters:
682   *   status          - The error number
683   *   feedbackObject  - The feedback object for more info
684   *
685   * Returns:
686   *   None.
687   *
688   *****************************************************************/

690  static void REST_ProgressDisplayError (eerrno_ty           status,
691                                         feedbackObjectPtr
                                                    feedbackObject)
692  {
693      Char            drawString[256];
694      Rect16Rec       wgtRect;
695      Rect16Rec       textRect;
696      Rect16Ptr       origBox;
697      Rect16Ptr       winBox;
698      EDMProgressPtr  edmObject;

700      if (REST_ProgressDisplayed)
701      {
702        if (status != E_SUCCESS)
703        {
704          if (((feedbackObject != NULL) &&
705               ((edmObject = EDMRST_GetFirstEDMObject (GREST_Handle,
706                                                       feedbackObject)) !=
                                                                 NULL))

707          {
708            TED_SetStrf ((TEdPtr)REST_ProgressWindow->StatusTed,
709                         "\n%s\n",
710                         e_get_error_text(status));
711          }
712          else
713          {
714            TED_SetStr ((TEdPtr)REST_ProgressWindow->StatusTed,
715                        e_get_error_text(status));
716          }
717        }

719        if (status == E_SUCCESS)
720        {
721          WIN_SetLabel ((WinPtr)REST_ProgressWindow,
722                        REST_GetErrorString (REST_SUCCESS_INDEX));
723          STR_Cpy (drawString, REST_GetErrorString (REST_RESTORE_SUCCESS));
724        }
725        else
726        {
727          WIN_SetLabel ((WinPtr)REST_ProgressWindow,
728                        REST_GetErrorString (REST_RESTORE_FAILURE));
729          STR_Cpy (drawString, e_get_error_text(status));
730        }

732        GUTIL_AddHostToWindowTitle ((WinPtr)REST_ProgressWindow);

734        /* Set the error as the client data */
735        WGT_SetClientRes ((WgtPtr)REST_ProgressWindow->ProgressArea,
736                          (ClientPtr)status);
```



```
738        WGT_Inval ((WgtPtr)REST_ProgressWindow->ProgressArea, BOOL_FALSE);

740        GUTIL_WGT_SetEnabled ((
                          WgtPtr)REST_ProgressWindow->CancelTBut, BOOL_TRUE);
741        TBUT_SetLabel ((TButPtr)REST_ProgressWindow->CancelTBut, "&OK");
742      }
743      else
744      {
745        if (status == E_SUCCESS)
746        {
747          /* Tell the user that the restoral completed successfully */
748          GALERT_DisplayError ((WinPtr)REST_RestoreWin,
749                               REST_GetErrorString (REST_SUCCESS_INDEX),
750                               GICON_GetIcon (I_SUCCESS),
751                               REST_GetErrorString (REST_RESTORE_SUCCESS));
752        }
753        else
754        {
755          /* Display the error */
756          GALERT_DisplayError ((WinPtr)REST_RestoreWin,
757                               REST_GetErrorString (REST_RESTORE_FAILURE),
758                               GICON_GetError(),
759                               e_get_error_text(status));
760        }

763    }
764  }
```

```
/*****************************************************************
 * Function Name:   REST_ProgressRemove ()
 *
 * Description:
 *   This routine will remove the progress window.
 *
 * Parameters:
 *   None.
 *
 * Success Outputs and Side Effects:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_ProgressRemove (void)
{
  /* We only care if the window is displayed */
  if (REST_ProgressDisplayed)
  {
    /* Reset the flags */
    REST_ProgressDisplayed = BOOL_FALSE;
    restoreInProgress = BOOL_FALSE;

    /* Remove the window */
    WIN_Terminate ((WinPtr)REST_ProgressWindow);
    REST_ProgressWindow = NULL;
  }
}
```



```
/*****************************************************************
 * REST_ProgressCancel
 *
 * Description:
 *   This routine will signal the progress dialog that the cancel
 *   button was hit.
 *
 * Parameters:
 *   userRequest - Flag if the user requested the cancel
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void  REST_ProgressCancel (BoolEnum userRequest)
{
  if (restoreInProgress)
  {
    if (!userRequest ||
        (GALERT_DisplayQuestion((WinPtr)REST_RestoreWin,
                                REST_GetErrorString(
                                        REST_VERIFY_CANCEL_TITLE),
                                GICON_GetQuestion(),
                                REST_GetErrorString(
                                        REST_VERIFY_CANCEL_MESSAGE),
                                BOOL_FALSE) == GALERT_Affirmative))

    {
      /*
       * tho user cancelled, display a cancel in progress dialog
       */

      restoreCancelled = BOOL_TRUE;

      WIN_SetLabel ((WinPtr)REST_ProgressWindow,
                    REST_GetErrorString (REST_CANCEL_TITLE));
      GUTIL_AddHostToWindowTitle ((WinPtr)REST_ProgressWindow);

      TED_SetStr ((TEdPtr)REST_ProgressWindow->StatusTed,
                  REST_GetErrorString (REST_CANCEL_PROGRESS));
      WGT_Inval ((
                  WgtPtr)REST_ProgressWindow->ProgressArea, BOOL_FALSE);

      GUTIL_WGT_SetEnabled((
                  WgtPtr)REST_ProgressWindow->CancelTBut, BOOL_FALSE);
    }
  }
  else
  {
    REST_ProgressRemove ();
  }
}
```

```
847  /*****************************************************************
848   * REST_DisplayDone
849   *
850   * Description:
851   *    This routine is the routine which will display the done
852   *    dialog with status for a file system restore.
853   *
854   * Parameters:
855   *    status          (I) - The restore status.
856   *    feedbackObject (I) - The restore feedback object
857   *
858   * Returns:
859   *    None.
860   *
861   *****************************************************************/

863  void REST_DisplayDone (eerrno_ty          status,
864                         feedbackObjectPtr feedbackObject)


866  1 {
867  1   EDMProgressPtr edmObject;
868  1   WIProgressPtr  wiObject;
869  1   Char           outString[MEDIUM_STRING_LENGTH];
870  1   time_t         now;

872  1   /* Reset the restore flags */
873  1   restoreInProgress = BOOL_FALSE;
874  1   restoreCancelled = BOOL_FALSE;

876  1   /* Update the end time to the current time */
877  1   now = time((time_t *)NULL);
878  1   strftime (outString,
879  1             MEDIUM_STRING_LENGTH,
880  1             "%H:%M",
881  1             localtime (&now));
882  1   TED_SetStr ((TEdPtr)REST_ProgressWindow->EndTimeTEd, outString);

884  1   /* If the status is successful, check for underlying errors */
885  1   if (status == E_SUCCESS)
886  2   {
887  2     edmObject = EDMRST_GetFirstEDMObject (
                                          GREST_Handle, feedbackObject);
888  2     if (EDMRST_GetObjectEDMFailed (GREST_Handle, edmObject) > 0)
889  3     {
890  3       wiObject = EDMRST_GetFirstObject (GREST_Handle, feedbackObject);
891  3       while ((status == E_SUCCESS) && (wiObject != NULL))
892  4       {
893  4         status = EDMRST_GetObjectItemStatus (GREST_Handle, wiObject);
894  4         wiObject = EDMRST_GetNextObject (GREST_Handle, wiObject);
895  3       }
896  2     }
897  1   }

899  1   /* Display the return status */
900  1   REST_ProgressDisplayError (status, feedbackObject);

902  1   REST_SetRestoreVisibility (BOOL_TRUE);
903  }
```



```
905  /*****************************************************************
906   * REST_RestoreInProgress
907   *
908   * Description:
909   *    This routine returns whether or not a restoral is currently in
                                                              progress.
910   *
911   * Parameters:
912   *    None.
913   *
914   * Returns:
915   *    BOOL_TRUE  - If a restoral has been started (
                                                    and is not yet finished)
916   *    BOOL_FALSE - Otherwise.
917   *
918   *****************************************************************/

920  BoolEnum  REST_RestoreInProgress (void)
921  1 {
922  1   return (restoreInProgress);
923  }
```

```
/*****************************************************************
 * REST_DisplayQuestion
 *
 * Description:
 *   This routine will display a question needed by the running
 *   restoral to the user.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

static void REST_DisplayQuestion (void)
{
  eerrno_ty                 status;
  queryObjectPtr            queryObject;
  int                       questionType;
  GPANEL_FieldInfo          *panelFields;
  int                       i;
  long                      cookie = INIT_COOKIE;
  int                       numAnswers = 0;
  Str                       *userAnswers = NULL;
  GPANEL_PanelIndexType panelType;
  int                       isDefault;
  int                       numChoices;
  int                       minLen;
  GPANEL_PanelHandle        panHandle;
  Char                      tempString[16];
  int                       selections;
  int                       selNumber;
  static BoolEnum           questionDisplayed = BOOL_FALSE;
  Char                      outString[64];

  if (!questionDisplayed)
  {

    if ((status = EDMRST_AllocQueryObject (GREST_Handle,
                                           &queryObject)) !=
                                                        E_SUCCESS)
    {
      REST_DisplayErrnoMessage ((WinPtr)REST_ProgressWindow,
                                NULL,
                                NULL,
                                status);

      return;
    }

    /* Flag that we are displaying a question dialog already */
    questionDisplayed = BOOL_TRUE;

    if (EDMRST_GetQuestion (GREST_Handle, queryObject) == E_SUCCESS)
    {
      questionType = EDMRST_GetQuestionType (
                                          GREST_Handle, queryObject);

      if (questionType == QTYPE_YESNO)
      {
        numAnswers = 1;
        userAnswers = GUTIL_Calloc (sizeof(Str));
        if (GALERT_DisplayQuestion
                ((WinPtr)REST_RestoreWin,
                 (Str)EDMRST_GetQuestionHeaderText (
```



```
                                          GREST_Handle, queryObject),
                 GICON_GetQuestion(),
                 (Str)EDMRST_GetQuestionText (
                                          GREST_Handle, queryObject),
                 BOOL_FALSE) == GALERT_Affirmative)
        {
          userAnswers[0] = esl_strdup ("Yes");
        }
        else
        {
          userAnswers[0] = esl_strdup ("No");
        }
      }
      else
      {

        switch (questionType)
        {
          case QTYPE_BOOL:
            panelType = GPANEL_1RADIO;
            panHandle = GPANEL_CreateHandle (panelType);
            panelFields = GPANEL_GetFields (panHandle);
            panelFields[0].details.radio.numChoices = 2;
            panelFields[0].details.radio.choices = GUTIL_Calloc (
                                                  2 * sizeof(Str));
            panelFields[0].details.radio.choices[0] = esl_strdup (
                                                        "True");
            panelFields[0].details.radio.choices[1] = esl_strdup (
                                                        "False");
            panelFields[0].details.radio.selection = 0;
            break;

          case QTYPE_RAD:
            panelType = GPANEL_1RADIO;
            panHandle = GPANEL_CreateHandle (panelType);
            panelFields = GPANEL_GetFields (panHandle);
            panelFields[0].details.radio.numChoices =
                                  EDMRST_GetQuestionNumChoices (
                                                  GREST_Handle,

                                                  queryObject);

            /* ONLY ALLOW THE MAXIMUM NUMBER OF RADIO CHOICES */
            if (
            panelFields[0].details.radio.numChoices > GPANEL_MAX_RBUT)
            {
              printf (
"Unsupported number of choices in radio box: %d. Can only accept %d.
                                                            \n",
                      panelFields[0].details.radio.numChoices,
                      GPANEL_MAX_RBUT);
              panelFields[0].details.radio.
                                          numChoices = GPANEL_MAX_RBUT;
            }

            panelFields[0].details.radio.choices =
              GUTIL_Calloc (
                panelFields[0].details.radio.numChoices * sizeof(Str));
            for (i=0; i < panelFields->details.radio.numChoices; i++)
            {
              panelFields[0].details.radio.choices[i] =
                esl_strdup (EDMRST_GetQuestionNextChoice (
                                                  GREST_Handle,
                                                  queryObject,
                                                  &isDefault,
                                                  &cookie));
```

```
1041 6              if (isDefault)
1042 7              {
1043 7                panelFields[0].details.radio.selection = i;
1044 6              }
1045 5            }
1046 5            break;

1048 5          case QTYPE_MULTI:
1049 5            numChoices = EDMRST_GetQuestionNumChoices (
                                                    GREST_Handle, queryObject);

1050 5            if (numChoices > 3)
1051 6            {
1052 6              printf (
                  "Unsupported number of choices: %d, can only accept 3\n",
                                numChoices);
1053 6              numChoices = 3;
1054 6            }
1055 5

1057 5            if (numChoices == 1)
1058 6            {
1059 6              panelType = GPANEL_1TOGGLE;
1060 5            }
1061 5            else if (numChoices == 2)
1062 6            {
1063 6              panelType = GPANEL_2TOGGLE;
1064 5            }
1065 5            else
1066 6            {
1067 6              panelType = GPANEL_3TOGGLE;
1068 5            }

1070 5            panHandle = GPANEL_CreateHandle (panelType);
1071 5            panelFields = GPANEL_GetFields (panHandle);

1073 5            for (i=0; i < numChoices; i++)
1074 6            {
1075 6              panelFields[i].fieldName = esl_strdup
1076 6                      (EDMRST_GetQuestionNextChoice (GREST_Handle,
1077 6                                                     queryObject,
1078 6                                                     &isDefault,
1079 6                                                     &cookie));
1080 6              panelFields[i].details.toggle.isSelected = isDefault;
1081 5            }
1082 5            break;

1084 5          case QTYPE_STR:
1085 5            panelType = GPANEL_1TSD;
1086 5            panHandle = GPANEL_CreateHandle (panelType);
1087 5            panelFields = GPANEL_GetFields (panHandle);
1088 5            panelFields[0].fieldName = NULL;
1089 5            panelFields[0].details.ted.isPassword = BOOL_FALSE;
1090 5            panelFields[0].details.ted.isOutputOnly = BOOL_FALSE;
1091 5            EDMRST_GetQuestionAnswerSize (GREST_Handle,
1092 5                                          queryObject,
1093 5                                          &minLen,
1094 5                                          &panelFields[0].details.ted.
                                                                  maxLength);

1095 5            panelFields[0].details.ted.invalidChars =
1096 5                    esl_strdup (EDMRST_GetQuestionInvalidChars (
                                                    GREST_Handle,
1097 5                                                queryObject));
1098 5            panelFields[0].details.ted.value = NULL;
1099 5            break;

1101 5          case QTYPE_INT:
```



```
1102 5            panelType = GPANEL_1SPIN;
1103 5            panHandle = GPANEL_CreateHandle (panelType);
1104 5            panelFields = GPANEL_GetFields (panHandle);
1105 5            panelFields[0].fieldName = NULL;
1106 5            panelFields[0].details.spin.value = 0;
1107 5            break;

1109 5          default:
1110 5            STR_Sprintf (outString,
1111 5                         "Unsupported question type %d.
                                                \nContact customer support.",
1112 5                         questionType);
1113 5            GALERT_DisplayError ((WinPtr)REST_RestoreWin,
1114 5                                 "Fatal Error",
1115 5                                 GICON_GetError(),
1116 5                                 outString);
1117 5            questionDisplayed = BOOL_FALSE;
1118 5            REST_ProgressCancel (BOOL_FALSE);
1119 5            EDMRST_FreeQueryObject (GREST_Handle, &queryObject);
1120 5            return;
1121 4        }

1123 4        /* Now create the panel with the appropriate question */
1124 4        GPANEL_CreatePanel (panHandle,
1125 4                            NULL,
1126 4                            (Str)EDMRST_GetQuestionText (GREST_Handle,
                                                                 queryObject),
1127 4                            NULL);
1128 4

1130 4        /* Create the window to display the panel in */
1131 4        REST_QuestionShow ((WinPtr)REST_RestoreWin,
1132 4                           queryObject,
1133 4                           panHandle,
1134 4                           panelFields);

1136 4        /* OK, now we have the answers,
                                        so fill in the answer structure */
1137 4        switch (questionType)
1138 5        {
1139 5          case QTYPE_BOOL:
1140 5            numAnswers = 1;
1141 5            userAnswers = GUTIL_Calloc (sizeof(Str));
1142 5            if (panelFields[0].details.radio.selection == 1)
1143 6            {
1144 6              userAnswers[0] = esl_strdup ("True");
1145 5            }
1146 5            else
1147 6            {
1148 6              userAnswers[0] = esl_strdup ("False");
1149 5            }
1150 5            break;

1152 5          case QTYPE_RAD:
1153 5            numAnswers = 1;
1154 5            userAnswers = GUTIL_Calloc (sizeof(Str));
1155 5            userAnswers[0] = esl_strdup (
    panelFields[0].details.radio.choices[panelFields[0].details.radio.
                                                              selection]);
1156 5            break;

1158 5          case QTYPE_MULTI:
1159 5            /* Count the number of selections */
1160 5            numAnswers = 0;
1161 5            for (i=0; i < numChoices; i++)
1162 6            {
1163 6              if (panelFields[i].details.toggle.isSelected)
```

```
                numAnswers++;
              }

              /* If no selections, send NULL as the answer */
              if (numAnswers == 0)
              {
                numAnswers = 1;
                userAnswers = GUTIL_Calloc (sizeof(Str));
                userAnswers[0] = NULL;
              }
              else
              {
                userAnswers = GUTIL_Calloc (selections * sizeof(Str));
                for (i=0; i < numChoices; i++)
                {
                  if (panelFields[i].details.toggle.isSelected)
                  {
                    userAnswers[selNumber] = esl_strdup (
                                                panelFields[i].fieldName);
                    selNumber++;
                  }
                }
              }
              break;

            case QTYPE_STR:
              numAnswers = 1;
              userAnswers = GUTIL_Calloc (sizeof(Str));
              if (panelFields[0].details.ted.value != NULL)
                userAnswers[0] = esl_strdup (
                                      panelFields[0].details.ted.value);
              else
                userAnswers[0] = NULL;
              break;

            case QTYPE_INT:
              numAnswers = 1;
              userAnswers = GUTIL_Calloc (sizeof(Str));
              sprintf (
                    tempString, "%d", panelFields[0].details.spin.value);
              userAnswers[0] = esl_strdup (tempString);
              break;

            default:
              break;
          }

          /* Now, clean up the memory for the generic panel */
          GPANEL_DestroyHandle (&panHandle, BOOL_TRUE);
        }

        /* Now give the restore API the answer(s) */
        for (i=0; i < numAnswers; i++)
        {
          EDMRST_SetUserAnswer (GREST_Handle,
                                queryObject,
                                userAnswers[i],
                                i < (numAnswers - 1));

          /* Done with the string for this answer, so free the memory */
          GUTIL_Free (userAnswers[i]);
        }

        /* Free the memory for the answer array */
        GUTIL_Free (userAnswers);
      }
```



```
    EDMRST_FreeQueryObject (GREST_Handle, &queryObject);

    /* Flag that the question dialog is no longer displayed */
    questionDisplayed = BOOL_FALSE;
  }
}
```

```
/*****************************************************************
 * REST_CheckForCancel
 *
 * Description:
 *   This routine will check to see if the restore has been cancelled
 *   by the user.
 *
 * Parameters:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_CheckForCancel (ClientPtr clientData)
{
  eerrno_ty               status;
                                                  /* Error Status */

  feedbackObjectPtr       feedbackObject;
  static RERunningState   lastState = RE_STATE_TIMEOUT;
  RERunningState          currentState = 0;
  BoolEnum                done = BOOL_FALSE;
  NotifyObjectPtr         notifyObject;
  Str                     nextString;
  Char                    stateString[CMAX_COMMAND_LENGTH];
  Char                    separatorString[CMAX_COMMAND_LENGTH];
  Int                     count;
  Int                     i;

  /* We only care if a restore is in progress */
  if (restoreInProgress &&
      (REST_ProgressWindow != NULL) &&
      RES_IsInitialized ((ResPtr)REST_ProgressWindow))

  {

    if ((status = EDMRST_AllocFeedbackObject (GREST_Handle,
                                              &feedbackObject)) !=
                                                        E_SUCCESS)

    {
      REST_DisplayErrnoMessage ((WinPtr)REST_ProgressWindow,
                                NULL,
                                REST_GetErrorString (
                                                    REST_START_ERROR),
                                status);
      return;
    }

    status = EDMRST_GetRestoreFeedback (GREST_Handle,
                                        restoreCancelled,
                                        &currentState,
                                        feedbackObject);


    if (!restoreCancelled)
    {
      REST_ProgressUpdate (feedbackObject);
    }

    if (status != EP_RB_RECOVER_RPC_INCOMPLETE)
    {
      done = BOOL_TRUE;
      restoreCancelled = BOOL_FALSE;
    }
```



```
    if (currentState != lastState)
    {
      /* Save the current state */
      lastState = currentState;

      /* Get the string to display */
      STR_Cpy (stateString, EDMRST_GetFeedbackStatus (currentState));

      /* Create a separator string of the same length */
      count = STR_Len (stateString);
      for (i=0; i < count; i++)
      {
        separatorString[i] = SEPARATOR_CHAR;
      }
      separatorString[count] = '\0';

      /*
       * Display the new state string and the separator
       */

      GTED_AppendStr ((TEdPtr)REST_ProgressWindow->NotifyMTed, "\n");
      GTED_AppendStr ((
                       TEdPtr)REST_ProgressWindow->NotifyMTed, stateString);
      GTED_AppendStr ((TEdPtr)REST_ProgressWindow->NotifyMTed, "\n");
      GTED_AppendStr ((
                       TEdPtr)REST_ProgressWindow->NotifyMTed, separatorString);
      GTED_AppendStr ((TEdPtr)REST_ProgressWindow->NotifyMTed, "\n");
    }

    notifyObject = EDMRST_GetFirstNotifyObject (
                                                GREST_Handle, feedbackObject);

    while (notifyObject != NULL)
    {
      nextString = (Str) EDMRST_GetNofifyMessageText (GREST_Handle,
                                                      notifyObject);

      GTED_AppendStr ((
                       TEdPtr)REST_ProgressWindow->NotifyMTed, nextString);
      GTED_AppendStr ((TEdPtr)REST_ProgressWindow->NotifyMTed, "\n");
      notifyObject = EDMRST_GetNextNotifyObject (
                                                 GREST_Handle, notifyObject);
    }

    /* Flush all events */
    EVENT_ProcessPending ();

    if (!done)
    {
      if (!REST_ProgressDisplayed ||
          ((int)WGT_GetClientRes (
                                  REST_ProgressWindow->ProgressArea) < 0))

      {
        EVENT_StartCBackAlarm (REST_CheckForCancel,
                               clientData,
                               RESTORE_CHECK_STARTUP_DELAY);

      }
      else
      {
        EVENT_StartCBackAlarm (REST_CheckForCancel,
                               clientData,
                               RESTORE_CHECK_CANCEL_DELAY);

      }

      if (currentState == RE_STATE_STOPPED)
      {
        REST_DisplayQuestion ();
      }
```

```
1357 2      }
1358 2      else
1359 1      {
1360 1        /* Display the Done dialog */
1361 3        REST_DisplayDone (status, feedbackObject);
1362 2      }

1364 2      /* Done with the feedback object */
1365 2      EDMRST_FreeFeedbackObject (GREST_Handle, &feedbackObject);
1366 1    }
1367      }
```



```
1369  /*********************************************************************
1370   * REST_GetSubmitResults
1371   *
1372   * Description:
1373   *   This routine get the results fo the submit call and starts
1374   *   the restore if successful.
1375   *
1376   * Parameters:
1377   *   clientData - The submit ID
1378   *
1379   * Returns:
1380   *   None.
1381   *
1382   *********************************************************************/

1384  static void REST_GetSubmitResults (ClientPtr  clientData)
1385 1 {
1386 1   unsigned int              submitId;
1387 1   eerrno_cy                 status;
1388 1   unsigned long             objectsDone = 0;

1390 1   submitId = (unsigned int) clientData;

1392 1   /* Get the results of the submit */
1393 3   if ((status = EDMRST_GetSubmitResults (GREST_Handle,
1394 1                                          restoreCancelled,
1395 1                                          &submitId,
1396 1                                          &objectsDone)) ==
                                               EP_RB_RECOVER_RPC_INCOMPLETE)

1397 2   {
1398 2     if (!restoreCancelled)
1399 3     {
1400 3       /* Update progress */
1401 3       REST_ProgressUpdate (NULL);
1402 2     }

1404 2     /* Add a timeout to try again after a delay */
1405 2     EVENT_StartCbackAlarm (REST_GetSubmitResults,
1406 2                            (ClientPtr)submitId,
1407 2                            RESTORE_CHECK_SUBMIT_DELAY);
1408 1   }
1409 1   else
1410 2   {

1412 2     /* If submit did not succeed, display an error message */
1413 2     if (status != E_SUCCESS)
1414 3     {
1415 3       restoreInProgress = BOOL_FALSE;
1416 3       restoreCancelled = BOOL_FALSE;
1417 3       REST_SetRestoreVisibility (BOOL_TRUE);

1419 3       REST_ProgressDisplayError (status, NULL);
1420 2     }
1421 2     else if (!restoreCancelled)
1422 3     {
1423 3       /* Gentlemen start your engines! */
1424 3       if ((status = EDMRST_Start (
                                        GREST_Handle, submitId)) != E_SUCCESS)

1425 4       {
1426 4         /* Er uh, won't start, display the error message */
1427 4         restoreInProgress = BOOL_FALSE;
1428 4         REST_SetRestoreVisibility (BOOL_TRUE);
1429 4         REST_ProgressDisplayError (status, NULL);
1430 3       }
1431 3       else
```

```
1432  4         {
1433  4           /* Call the check for cancel routine to display the progress
                                                                    dialog */
1434  4           REST_CheckForCancel ((ClientPtr)&checkForCancelId);
1435  3         }
1436  2       }
1437  1     }
1438    }
```



```
1440  /*************************************************************************
1441   * REST_StartProgress
1442   *
1443   * Description:
1444   *    This routine will start the restore process and monitor its
                                                                    progres.
1445   *
1446   * Parameters:
1447   *    None.
1448   *
1449   * Returns:
1450   *    None.
1451   *
1452   *************************************************************************/

1454  void REST_StartProgress ( void )
1455  1 {
1456  1   REST_SetRestoreVisibility (BOOL_FALSE);

1458  1   REST_FirstDataTime = 0;

1460  1   REST_ProgressDisplay ();

1462  1   restoreInProgress = BOOL_TRUE;

1464  1   /* Add a timeout to get the submit results */
1465  1   EVENT_StartCBackAlarm (REST_GetSubmitResults,
1466  1                          (ClientPtr)0,
1467  1                          RESTORE_CHECK_SUBMIT_DELAY);
1468    }
```

```
/* -- Template created by NEURON DATA Open Interface.                    */
/* -- Do not alter 'CodeGen' directives.                                 */
/* {{ CodeGen: GeneratorVersion 4 }}                                     */
/* -- Code generated on 06/30/99 at 10:43:40.                           */
/* -- Code regenerated on 07/22/99 at 13:35:35.                         */
/* -- Code regenerated on 07/22/99 at 13:36:30.                         */
/* -- Code regenerated on 08/12/99 at 14:39:17.                         */
/* -- Code regenerated on 08/13/99 at 10:57:20.                         */
/* -- Code regenerated on 08/16/99 at 10:34:32.                         */
/* -- Code regenerated on 08/30/99 at 15:12:28.                         */
/* -- Code regenerated on 08/31/99 at 07:58:51.                         */
/* -- Code regenerated on 08/31/99 at 08:02:13.                         */
/* -- Code regenerated on 08/31/99 at 08:10:32.                         */
/* -- Code regenerated on 11/10/99 at 15:59:13.                         */
/* {{ CodeGen: CodeHistory }}                                            */

#define ERR_LIB RESTORE

#include "restProgress.h"
#include "gutil/guiUtils.h"
#include "gutil/tcdUtils.h"
#include "gutil/winUtils.h"

ERR_EXTERN
ERR_INMODULE("RestProgress")

/* {{ CodeGen: ClassImplementationPlaceHolder }}                         */
/* {{ CodeGen: WinClassImplementationPlaceHolder }}                      */

/* {{ CodeGen: WindowSection Win                                         */
/* ======================================================================= */
/* -- Code for Window "Win"                                              */
/* ======================================================================= */

/* {{ CodeGen: MenuImplementationPlaceHolder }}                          */

/* {{ CodeGen: WgtNfyHandler ValidateWiNameTEd                           */
static   void C_FAR RestProgressWin_ValidateWiNameTEd L1(
                                                RestProgressWinPtr, win)
{
}
/* }} CodeGen: WgtNfyHandler ValidateWiNameTEd
```



```
                                                                      */
/* {{ CodeGen: WgtNfyHandler HitCancelTBut                               */
static   void C_FAR RestProgressWin_HitCancelTBut L1(
                                                RestProgressWinPtr, win)
{
  REST_ProgressCancel (BOOL_TRUE);
}
/* }} CodeGen: WgtNfyHandler HitCancelTBut                               */

/* {{ CodeGen: WgtNfyHandler ValidateStartTimeTEd                        */
static   void C_FAR RestProgressWin_ValidateStartTimeTEd L1(
                                                RestProgressWinPtr, win)
{
}
/* }} CodeGen: WgtNfyHandler ValidateStartTimeTEd
                                                                      */
/* {{ CodeGen: WgtNfyHandler ValidateEndTimeTEd                          */
static   void C_FAR RestProgressWin_ValidateEndTimeTEd L1(
                                                RestProgressWinPtr, win)
{
}
/* }} CodeGen: WgtNfyHandler ValidateEndTimeTEd
                                                                      */
/* {{ CodeGen: WgtNfyHandler ValidateStatusTEd                           */
static   void C_FAR RestProgressWin_ValidateStatusTEd L1(
                                                RestProgressWinPtr, win)
{
}
/* }} CodeGen: WgtNfyHandler ValidateStatusTEd
                                                                      */
/* {{ CodeGen: WgtNfyHandler ValidateNotifyMTEd                          */
static   void C_FAR RestProgressWin_ValidateNotifyMTEd L1(
                                                RestProgressWinPtr, win)
{
}
/* }} CodeGen: WgtNfyHandler ValidateNotifyMTEd
                                                                      */
/* {{ CodeGen: WgtNfyPlaceHolder }}                                      */

/* {{ CodeGen: UseDefaultNfyHandler name_of_nfy_handler }}               */
/* {{ CodeGen: UseAllDefaultNfyHandlers name_of_wgt_member }}            */

static  void RestProgressWin_RedrawProgressArea (
                                                RestProgressWinPtr win)
{
  WGT_DefNfy ((WgtPtr)win->ProgressArea, WGT_NFYREDRAW);
  REST_ProgressDrawPercentComplete ((WgtPtr)win->ProgressArea);
}

void     RestProgressWin_Construct L1(RestProgressWinPtr, win)
{
        /* {{
        CodeGen: WgtInitializations                                   */
```

```
88 1            win->ProgressPanel = (PanelPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "ProgressPanel");
89 1            win->WiNameTArea = (TAreaPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "WiNameTArea");
90 1            win->WiNameTEd = (STEdPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "WiNameTEd");
91 1            win->ProgressArea = (PanelPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "ProgressArea");
92 1            win->StartTimeTEd = (STEdPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "StartTimeTEd");
93 1            win->EndTimeTEd = (STEdPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "EndTimeTEd");
94 1            win->StatusTed = (MTEdPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "StatusTed");
95 1            win->NotifyMTed = (MTEdPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "NotifyMTed");
96 1            win->CancelTBut = (PButPtr)PANEL_GetNamedWgt((
                                                PanelPtr)win, "CancelTBut");
97 1            WIN_SetWgtNfyHandler((WinPtr)win, (
                                                WgtPtr)win->WiNameTEd, TED_NFYVALIDATE,
98 1                    (
                    WinWgtNfyHandlerProc)RestProgressWin_ValidateWiNameTEd);
99 1            WIN_SetWgtNfyHandler((WinPtr)win, (
                                                WgtPtr)win->StartTimeTEd, TED_NFYVALIDATE,
100 1                   (
                WinWgtNfyHandlerProc)RestProgressWin_ValidateStartTimeTEd);
101 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                                WgtPtr)win->EndTimeTEd, TED_NFYVALIDATE,
102 1                   (
                    WinWgtNfyHandlerProc)RestProgressWin_ValidateEndTimeTEd);
103 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                                WgtPtr)win->StatusTed, TED_NFYVALIDATE,
104 1                   (
                 WinWgtNfyHandlerProc)RestProgressWin_ValidateStatusTed);
105 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                                WgtPtr)win->NotifyMTed, TED_NFYVALIDATE,
106 1                   (
                WinWgtNfyHandlerProc)RestProgressWin_ValidateNotifyMTed);
107 1           WIN_SetWgtNfyHandler((WinPtr)win, (
                                                WgtPtr)win->CancelTBut, TBUT_NFYHIT,
108 1                   (WinWgtNfyHandlerProc)RestProgressWin_HitCancelTBut);
109 1           /* )) CodeGen: WgtInitializations 129970
                                                */

111 1   WGT_SetNfyProc ((WgtPtr)win->WiNameTEd,    GUTIL_OutputTED_DefNfy);
112 1   WGT_SetNfyProc ((WgtPtr)win->StartTimeTEd, GUTIL_OutputTED_DefNfy);
113 1   WGT_SetNfyProc ((WgtPtr)win->EndTimeTEd,   GUTIL_OutputTED_DefNfy);
114 1   WGT_SetNfyProc ((WgtPtr)win->StatusTed,    GUTIL_OutputTED_DefNfy);
115 1   WGT_SetNfyProc ((WgtPtr)win->NotifyMTed,   GUTIL_OutputTED_DefNfy);

117 1   WIN_SetWgtNfyHandler((WinPtr)win, (WgtPtr)win, WGT_NFYRESIZE,
118 1                       (WinWgtNfyHandlerProc)GUTIL_WinHandleResize);
119 1   WIN_SetWgtNfyHandler((WinPtr)win, (WgtPtr)win, WIN_NFYMOUSECLICK,
120 1                       (
                        WinWgtNfyHandlerProc)GUTIL_WinHandleMouseClick);

122 1           WIN_SetWgtNfyHandler((WinPtr)win,
123 1                               (WgtPtr)win->ProgressArea,
124 1                               PANEL_NFYREDRAW,
125 1                               (
                    WinWgtNfyHandlerProc)RestProgressWin_RedrawProgressArea);


127 }

129 RestProgressWinPtr          REST_ProgressWindowInit (WinPtr parent)
130 1 {
```



```
131 1           RestProgressWinPtr win;

133 1           win = (RestProgressWinPtr)WIN_LoadSized(
                                                "RestProgress", "Win",
134 1                   sizeof(RestProgressWinRec));
135 1           RestProgressWin_Construct(win);

137 1   /* Set this module to be non-visible when closed */
138 1   if (parent != NULL)
139 2   {
140 2     WIN_SetParentWin ((WinPtr)win, parent);
141 2     WIN_SetOptFlags  ((WinPtr)win, WIN_OPTABOVEPARENT);
142 2     WIN_SetIconFlags ((WinPtr)win, WIN_OPTABOVEPARENT);
143 1   }

145 1           WIN_Init((WinPtr)win);

147 1   return (win);
148   }

150   /* )) CodeGen: WindowSection Win                    */

151   /* {{ CodeGen: WindowImplementationPlaceHolder }}   */

153   /* {{ CodeGen: MainPlaceHolder }}                   */
```

```
/*****************************************************************
 * restSearchMgr.c
 *
 *
 * Copyright 1996 by Epoch Systems, Inc.
 *
 *
 * Mission Statement:
 *   This file contains the functions necessary for the EDM Restore
 *   Search window.
                Note that searching only works for file system workitems,
 *   no OLDB support is included.

 *
 * Required Includes:
 *   None
 *
 * Compile-Time Options:
 *   N/A
 *

 *
 * RCS Information:
 *      $RCSfile$
 *      $Revision$
 *      $Date$
 *****************************************************************/

#define  ERR_LIB            RESTORE

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>

#include <respub.h>
#include <cboxpub.h>
#include <panelpub.h>
#include <tbutpub.h>
#include <winpub.h>
#include <drawpub.h>

#include "eerrno/e_errno.h"
#include "util/esl_string.h"
#include <restore/restore_api.h>

#include "restore.h"
#include "restoreP.h"
#include "restUtils.h"
#include "restSolMgr.h"
#include "restCalendar.h"
#include "restSearch.h"
#include "restAPIUtils.h"
#include "gutil/timeUtils.h"
#include "gutil/alertMgr.h"
#include "gutil/iconDefs.h"
#include "gutil/iconUtils.h"
#include "gutil/winUtils.h"
#include "gutil/lboxUtils.h"
#include "gutil/guiUtils.h"

/*************
 * Constants *
 *************/

#define MAX_FOUND_OBJECTS     10    /* Maximum objects to find per
                                                        iteration */
#define NUMBER_FOUND_COLUMNS 10     /* Number of columns in the list box
                                                                  */
```



```
#define BACKUP_TIME_COLUMN 1
#define MARK_COLUMN        2
#define ICON_COLUMN        3
#define NAME_COLUMN        4
#define PERMISSIONS_COLUMN 5
#define OWNER_COLUMN       6
#define GROUP_COLUMN       7
#define SIZE_COLUMN        8
#define DATE_COLUMN        9
#define TIME_COLUMN        10

/*
 * Number of milliseconds to delay
 */

#define SEARCH_ALARM_DELAY              100
#define RESTORE_START_FIND_RESULTS_DELAY 500
#define RESTORE_CONT_FIND_RESULTS_DELAY   10

/*************************
 * Local Data structures *
 *************************/

typedef struct _REST_FoundObjectRec
{
   time_t        backupTime;
                                /* The time of the backup for this object */
   GREST_Object object;                 /* The object */
   struct _REST_FoundObjectRec *next;/* The next found object */
} REST_FoundObjectRec, *REST_FoundObjectPtr;

/********************
 * Global Variables *
 ********************/

/* Icons for objects */
static IconPtr searchDirIcon;
static IconPtr searchFileIcon;
static IconPtr searchCheckBoxIcon;
static IconPtr searchCheckIcon;
static IconPtr searchSelCheckIcon;
static IconPtr searchOffsiteIcon;
static IconPtr searchBadIcon;

/* First and last objects in the list */
static REST_FoundObjectPtr firstFoundObject = NULL;
static REST_FoundObjectPtr lastFoundObject = NULL;

/* Number of found entries */
static unsigned long REST_CurrentEntries = 0;

/* Flags for current status */
static BoolEnum REST_SearchTerminated = BOOL_FALSE;
static BoolEnum REST_SearchDisplayed = BOOL_FALSE;

/* Handle to the current search in progress dialog */
static GALERT_WinHandle synchSearchHandle = NULL;

/* Flags for search in progress */
static BoolEnum searchInProgress = BOOL_FALSE;

static long REST_SearchCookie = INIT_COOKIE;
                                                /* Ah, the magic cookie */

/*****************************************************************
```

```
 128   * Function Name:    REST_AddFoundItem ()
 129   *
 130   * Description:
 131   *   This routine will add a found object to the search results
 132   *   list box (at the end of the list box).
 133   *
 134   * Parameters:
 135   *   object      (I) - The object to add
 136   *   backupTime (I) - The backup time for the object
 137   *
 138   * Success Outputs and Side Effects:
 139   *   None.
 140   *
 141   * Returns:
 142   *   None.
 143   *
 144   ************************************************************/

 145   static void REST_AddFoundItem (GREST_Object object,
 147                                  time_t       backupTime)
 148 1 {
 149 1   REST_FoundObjectPtr foundObject;  /* New found object record */

 151 1   /* Create a new found object record */
 152 1   foundObject = (REST_FoundObjectPtr) GUTIL_Malloc(sizeof(
                                                 REST_FoundObjectRec));

 154 1   /* Fill in the fields */
 155 1   foundObject->backupTime = backupTime;
 156 1   foundObject->object = object;
 157 1   foundObject->next = NULL;

 159 1   /* Put this on the end of the list of found objects */
 160 1   if (lastFoundObject != NULL)
 161 2   {
 162 2     /* Tack it onto the end */
 163 2     lastFoundObject->next = foundObject;
 164 2     lastFoundObject = foundObject;
 165 1   }
 166 1   else
 167 2   {
 168 2     /* This is the only object, first and last */
 169 2     firstFoundObject = foundObject;
 170 2     lastFoundObject = foundObject;
 171 1   }
 172   }
```



```
 174   /************************************************************
 175    * REST_CheckForCancel
 176    *
 177    * Description:
 178    *   This routine will determine if the user has cancelled the search
                                                                    and
 179    *   will update the progress window.
 180    *
 181    * Parameters:
 182    *   None.
 183    *
 184    * Returns:
 185    *   BOOL_TRUE  - if the user has cancelled
 186    *   BOOL_FALSE - otherwise
 187    *
 188    ************************************************************/

 190   static BoolEnum REST_CheckForCancel (void)
 191 1 {
 192 1   static long lastNumFound = -1;
                                        /* Number of items found previously */
 193 1   Char        outputString[MAX_STRING_LENGTH];
                                        /* Updated string to display */
 194 1   BoolEnum    retVal;             /* Value to return */

 196 1   /* Make sure the cancel dialog is still displayed */
 197 1   if (synchSearchHandle != NULL)
 198 2   {
 199 2     /* If the number of entries found has changed,
                                                    update the string */
 200 2     if (REST_CurrentEntries != lastNumFound)
 201 3     {
 202 3       /* Build the new string */
 203 3       if (REST_CurrentEntries != 0)
 204 4       {
 205 4         STR_Sprintf (outputString,
 206 4                      REST_GetErrorString (REST_SEARCH_STATUS),
 207 4                      REST_CurrentEntries);
 208 4       }
 209 3       else
 210 4       {
 211 4         STR_Cpy (outputString, REST_GetErrorString (
                                                REST_SEARCH_IN_PROGRESS));
 212 3       }

 214 3       /* Update the progress dialog */
 215 3       GALERT_UpdateMessage (synchSearchHandle, outputString);

 217 3       /* Save the current number of entries */
 218 3       lastNumFound = REST_CurrentEntries;
 219 2     }

 221 2     /* Return whether or not the user cancelled */
 222 2     retVal = GALERT_IsCancelled(synchSearchHandle);
 223 1   }
 224 1   else
 225 2   {
 226 2     retVal = BOOL_TRUE;
 227 1   }

 229 1   return (retVal);
 230   }
```

```
232  /*************************************************************************
233   * REST_SearchStartTimeout
234   *
235   * Description:
236   *    This routine will start the search.
237   *                                        It is used to delay a bit to make
238   *    sure the in progress dialog is visible.
239   *
240   * Parameters:
241   *    clientData (I) - Unused.
242   *
243   * Returns:
244   *    None.
245   *
246   *************************************************************************/
247  static void REST_SearchStartTimeout (ClientPtr clientData)
248  {
249    REST_SearchStartSearch ();
250  }
```



```
252  /*************************************************************************
253   * REST_SearchCancel
254   *
255   * Description:
256   *    This routine will remove the search in progress dialog and clear
257   *    the current sync window handle.
258   *
259   * Parameters:
260   *    None.
261   *
262   * Returns:
263   *    None.
264   *
265   *************************************************************************/

267  void REST_SearchCancel (void)
268  {
269    long          numEntries;       /* Number found */
270    GREST_Object  foundObjects[1];  /* Temp array of found objects */
271    time_t        times[1];         /* Backup time for object */
272    eerrno_ty     eerrno;           /* Error code returned */

274    /* If there really is a search in progress, remove the dialog */
275    if (synchSearchHandle != NULL)
276    {
277      GALERT_CancelSynchDialog (synchSearchHandle);
278      synchSearchHandle = NULL;
279    }

281    /*
282     * If there is a search in progress then cancel the find operation
283     */

285    if (searchInProgress)
286    {
287      /* Cancel the find operation, ignore results */
288      EDMRST_GetFindResults (GREST_Handle,
289                             BOOL_TRUE,
290                             1,
291                             foundObjects,
292                             times,
293                             &numEntries,
294                             &REST_SearchCookie);
295    }
296  }
```

```
298   /*****************************************************************
299    * REST_DisplaySearchInProgress
300    *
301    * Description:
302    *   This routine will display the search in progress dialog and
303    *   initialize the current number of entries found.
304    *
305    * Parameters:
306    *   None.
307    *
308    * Returns:
309    *   None.
310    *
311    *****************************************************************/

313   void REST_DisplaySearchInProgress (void)
314 1 {
315 1   time_t  startTime;  /* Current start time */
316 1   time_t  endTime;    /* Current end time */

318 1   /* Do nothing if we're already searching,
                                          or a restore is in progress */
319 1   if (REST_SearchInProgress() || REST_RestoreInProgress())
320 1     return;

322 1   /* Get the start backup date/time */
323 1   startTime = (time_t)WGT_GetClientRes ((
                              WgtPtr)REST_SearchWindow->StartDateOutput);

325 1   /* Get the end backup date/time */
326 1   endTime = (time_t)WGT_GetClientRes ((
                              WgtPtr)REST_SearchWindow->EndDateOutput);


328 1   /* Validate the search start and end times */
329 1   if (startTime > endTime)
330 2   {
331 2     /* The start time is after the end time,
                                          no searching can be done */
332 2     GALERT_DisplayError ((WinPtr)REST_SearchWindow,
333 2                          REST_GetErrorString (REST_SEARCH_FAIL_TITLE),
334 2                          GICON_GetError(),
335 2                          REST_GetErrorString (REST_SEARCH_TIME_ERROR));
336 1   }
337 1   else
338 2   {

340 2     /* Initialize the number of current entries to zero */
341 2     REST_CurrentEntries = 0;

343 2     /* Initialize the window */
344 2     synchSearchHandle = GALERT_DisplaySynchronousWait
345 2                          ((WinPtr)REST_SearchWindow,
346 2                           REST_GetErrorString (
                                          REST_SEARCH_PROGRESS_TITLE),
347 2                           GICON_GetIcon (I_WAIT),
348 2                           REST_GetErrorString (
                                          REST_SEARCH_IN_PROGRESS),
349 2                           BOOL_TRUE);

351 2     /* Delay a bit to make sure the dialog is visible */
352 2     EVENT_StartCBackAlarm (REST_SearchStartTimeout,
353 2                            (ClientPtr)NULL,
354 2                            SEARCH_ALARM_DELAY);
355 1   }
```



```
357   }
```

```
 359 1  /*****************************************************************************
 360 1   * REST_SearchGetFoundItem
 361 1   *
 362 1   * Description:
 363 1   *   This routine will return the found item that would appear in the
 364 1   *   given row.
 365 1   *
 366 1   * Parameters:
 367 1   *   row (I) - The row to find the item in
 368 1   *
 369 1   * Returns:
 370 1   *   REST_FoundObjectPtr - item in row if row is valid
 371 1   *                       - NULL if invalid row,
 372 1   *                                          <= 0 or > number of rows
 373 1   *
 374 1   *****************************************************************************/

 375 1  static REST_FoundObjectPtr REST_SearchGetFoundItem (Int32 row)
 376 1  {
 377 1    REST_FoundObjectPtr foundObject = NULL;
 378 1    REST_FoundObjectPtr nextObject = NULL;
 379 1    Int32               count = 1;

 381 1    /* Validate the row first */
 382 1    if (row > 0)
 383 2    {
 384 2      /* get to the correct object in the list */
 385 2      nextObject = firstFoundObject;
 386 2      while ((nextObject != NULL) && (count < row))
 387 3      {
 388 3        nextObject = nextObject->next;
 389 3        count ++;
 390 2      }

 392 2      foundObject = nextObject;
 393 1    }

 395 1    /* Return the appropriate object */
 396 1    return (foundObject);
 397  }
```



```
 399  /*****************************************************************************
 400   * REST_SearchUpdateButtons
 401   *
 402   * Description:
 403   *   This routine will update the sensitivity of the search results
 404   *   action buttons (
                          mark, unmark, set view, clear) based on the current
 405   *   selections.
 406   *
 407   * Parameters:
 408   *   None
 409   *
 410   * Returns:
 411   *   None.
 412   *
 413   *****************************************************************************/

 415  void REST_SearchUpdateButtons (void)
 416  {
 417    Int32               foundRow;
                                        /* Current row selection */
 418    REST_FoundObjectPtr foundObject;
                                        /* Next object in the list */
 419    BoolEnum            itemsExist = BOOL_FALSE;
                                        /* Flag if there are items */
 420    int                 selectedCount = 0;
                                        /* Number of selected items */
 421    BoolEnum            unmarkEnabled = BOOL_FALSE;
                                        /* Flag to unable unmark */
 422    BoolEnum            markEnabled = BOOL_FALSE;
                                        /* Flag to unable mark */
 423    BoolEnum            setViewEnabled = BOOL_FALSE;
                                        /* Flag to unable set view */

 425    /* If a restore is in progress, don't update any sensitivity */
 426    if (REST_RestoreInProgress())
 427      return;

 429    /*
 430     * Loop through the list examining the selected items
 431     */

 433    /* Get the object at the first selected row position */
 434    foundRow = QUTIL_LBOX_GetFirstVLboxSelectedRow (
                                        REST_SearchWindow->FoundBox);
 435    foundObject = REST_SearchGetFoundItem (foundRow);

 437    while (foundObject != NULL)
 438    {
 439      /* At least we know there are entries in the list */
 440      itemsExist = BOOL_TRUE;

 442      /* Bump the selected count */
 443      selectedCount++;

 445      /* if the object is not marked, determine if it is markable */
 446      if (!GREST_IsObjectMarkedForTime (GREST_Handle,
 447                                        foundObject->object,
 448                                        foundObject->backupTime))
 449      {

 451        /* Determine if this object is markable */
 452        if (GREST_IsObjectMarkableForTime (GREST_Handle,
 453                                           foundObject->object,
 454                                           foundObject->backupTime) &&
```

```
            ((EDMRST_GetObjectStatus (GREST_Handle,
                                      foundObject->object) !=
                                                    Backup_Bad) ||
             REST_MarkBadFiles))
        {
          /* It is markable, so enable the mark button */
          markEnabled = BOOL_TRUE;
        }
      }

      /* Else this object is marked so enable the unmark button */
      else
      {
        unmarkEnabled = BOOL_TRUE;
      }

      /* Get the object at the next selected row position */
      foundRow = GUTIL_LBOX_GetNextVLboxSelectedRow (
                                                REST_SearchWindow->FoundBox);
      foundObject = REST_SearchGetFoundItem (foundRow);
    }

    /* If any items were in the list */
    if (itemsExist)
    {
      /* Only enable the set view button if only one object is selected
                                                                      */
      if (selectedCount == 1)
        setViewEnabled = BOOL_TRUE;
    }

    /* Apply what we have determined */
    WGT_SetEnabled ((WgtPtr)REST_SearchWindow->MarkButton, markEnabled);
    WGT_SetEnabled ((
                     WgtPtr)REST_SearchWindow->UnMarkButton, unmarkEnabled);
    WGT_SetEnabled ((
                     WgtPtr)REST_SearchWindow ->SetViewButton, setViewEnabled);
}
```



```
/*****************************************************************
 * REST_SearchFillLBox
 *
 * Description:
 *   This routine will fill the virtual listbox rows given the start
 *   row (number in row 1) and the bounds of the list box rows.
 *
 * Parameters:
 *   lbox      (I) - The virtual list box to fill
 *   startRow  (I) - The real row number that is now in row 1
 *   bounds    (I) - The number of rows in the list box
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

static void REST_SearchFillLBox (LBoxPtr  lbox,
                                 Int32    startRow,
                                 Int16    bounds)
{
  REST_FoundObjectPtr nextObject;
  ClientPtr           clientData;
  Int32               count = 1;
  Int16               row;

  /* get to the correct object in the list */
  nextObject = firstFoundObject;
  while ((nextObject != NULL) && (count < startRow))
  {
    nextObject = nextObject->next;
    count ++;
  }

  /* Put the next set of rows in the lbox, even if NULL */
  REST_SearchLBoxFrozen = BOOL_TRUE;
  row = 1;
  while (row <= bounds)
  {
    /* Go to the row (rows start at 1) */
    LBOX_GoCellRow (lbox, row);

    /* Set the client data for this row */
    LBOX_CurSetClientData (lbox, (ClientPtr)nextObject);

    /* go to the next object */
    if (nextObject != NULL)
      nextObject = nextObject->next;
    row++;
  }
  REST_SearchLBoxFrozen = BOOL_FALSE;


  /* Now fit in the client data */
  LBOX_GoHome (REST_SearchWindow->FoundBox);
  row = 1;
  while ((clientData = LBOX_CurGetClientData (
                                  REST_SearchWindow->FoundBox)) != NULL)
  {
    GUTIL_FitThisDataInRow (REST_SearchWindow->FoundBox,
                            clientData,
                            row,
                            NUMBER_FOUND_COLUMNS,
                            REST_GetSearchListColumnValues);
    row++;
```

```
    LBOX_GoCollRow (REST_SearchWindow->FoundBox, row);
  }
}
```



```
/*****************************************************************
 * REST_SearchDisplay
 *
 * Description:
 *   This routine will display the search window, with the starting
 *   search directory to the given directory (/ if NULL).
 *
 * Parameters:
 *   currentDirectory (I) - Directory to start the search from
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_SearchDisplay (Str currentDirectory)
{
  time_t          currentTime;
                                        /* Current backup time */
  Char            dateString[SMALL_STRING_LENGTH];
                                        /* String for the date */
  GREST_Object    currentWI;
                                        /* Work-item to search */
  Char            windowLabel[MAX_STRING_LENGTH];
                                        /* Lable for the window */
  Char            name[MAX_CLIENT_NAME_LENGTH];
                                        /* Name of the Work-Item */

  if (!REST_SearchDisplayed)
  {
    /* Load up the window's resources */
    REST_SearchWindowInit ();

    /* Set buttons and labels to default values */
    TED_SetStr ((
            TEdPtr)REST_SearchWindow->DirectoryText, currentDirectory);
    TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->DescendToggle, BOOL_TRUE);
    TED_ClearAll ((TEdPtr)REST_SearchWindow->StringText);
    TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->IncludeRadio, BOOL_TRUE);
    TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->ExcludeRadio, BOOL_FALSE);
    TED_ClearAll ((TEdPtr)REST_SearchWindow->OwnerText);
    TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->IncludeOwnerToggle, BOOL_TRUE);
    TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->ExcludeOwnerToggle, BOOL_FALSE);
    TED_ClearAll ((TEdPtr)REST_SearchWindow->GroupText);
    TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->IncludeGroupToggle, BOOL_TRUE);
    TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->ExcludeGroupToggle, BOOL_FALSE);
    TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->AllTypesToggle, BOOL_TRUE);
    TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->AllStatusToggle, BOOL_TRUE);
    TED_ClearAll ((TEdPtr)REST_SearchWindow->SizeText);
    TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->GreaterThanToggle, BOOL_FALSE);
    TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->LessThanToggle, BOOL_FALSE);
    TBUT_SetSelected ((
                    TButPtr)REST_SearchWindow->EqualsToggle, BOOL_TRUE);
```

```
604 2     /* Set the date strings to the current backup date */
605 2     if (EDMRST_GetCurrentBackupTime (GREST_Handle, &currentTime) == 0)
606 3     {
607 3        strftime (dateString,
                    SMALL_STRING_LENGTH,
                    "%b %d %H:%M",
                    localtime (&currentTime));
608 3     }
609 3     else
610 3     {
611 4        /* No current time (?), blank out the string */
612 3        STR_Cpy (dateString, "");
613 3     }
614 2
615 2     TED_SetStr ((
                    TEdPtr)REST_SearchWindow->StartDateOutput, dateString);
616 2     TED_SetStr ((TEdPtr)REST_SearchWindow->EndDateOutput, dateString);
617 2     WGT_SetClientRes ((WgtPtr)REST_SearchWindow->StartDateOutput, (
                                                   ClientPtr)currentTime);
618 2     WGT_SetClientRes ((WgtPtr)REST_SearchWindow->EndDateOutput, (
                                                   ClientPtr)currentTime);
619 2
620 2     /* Make the list box a virtual list box */
621 2     GUTIL_LBOX_SetVirtual (REST_SearchWindow->FoundBox,
                               REST_SearchWindow->FoundSbar,
                               24,
                               NULL,
                               REST_SearchFillLBox);
622 2
623 2     /* We ain't found nothing yet */
624 2     REST_ClearFoundBox ();
625 2
626 2     /* Set the status flag */
627 2     REST_SearchTerminated = BOOL_FALSE;
628 2
629 2     /* Set the label for the window */
630 2     currentWI = REST_GetCurrentWorkItem ();
631 2     STR_Cpy (name, EDMRST_GetObjectFullName (GREST_Handle, currentWI));
632 2     GUTIL_SetDefaultWindowTitle ((WinPtr)REST_SearchWindow, name);
633 2
634 2     REST_SearchDisplayed = BOOL_TRUE;
635 1  }
636 1
637 1  /* Put up the window */
638 1  WIN_Show ((WinPtr)REST_SearchWindow);
639
640 }
```



```
649  /*****************************************************************
650   * Function Name:   REST_SearchDisposeInfo ()
651   *
652   * Description:
653   *   This routine will free the info in the current cell.
654   *
655   * Parameters:
656   *   info (I) - The found object info pointer.
657   *
658   * Success Outputs and Side Effects:
659   *   None.
660   *
661   * Returns:
662   *   None.
663   *
664   *****************************************************************/
665
666  static void REST_SearchDisposeInfo (REST_FoundObjectPtr info)
667  {
668
669     /* Get the info */
670     if (info == NULL)
671     {
672        /* Free the restore object */
673        EDMRST_FreeRestorableObjects (GREST_Handle, &info->object, 1);
674
675        /* Free the info */
676        GUTIL_Free (info);
677     }
678  }
```

```
680
681   /*****************************************************************
682    * Function Name:    REST_SearchRemove ()
683    *
684    * Description:
685    *    This routine will remove the search window.
686    *
687    * Parameters:
688    *    None.
689    *
690    * Success Outputs and Side Effects:
691    *    None.
692    *
693    * Returns:
694    *    None.
695    *
696    *****************************************************************/
697
698 1 BoolEnum REST_SearchRemove (void)
699 1 {
700 1   BoolEnum              returnStatus;  /* Duh, status to remove */
701 1   REST_FoundObjectPtr thisObject;
                                  /* Pointer to found object to dispose */
702 1   REST_FoundObjectPtr nextObject;
                                  /* Next found object in the list */
703 1
704 1   /* Do nothing if a restore is in progress */
705 1   if (REST_RestoreInProgress())
706 2   {
707 2     return (BOOL_FALSE);
708 1   }
709 1
710 1   /* We only care if the window is displayed */
711 1   if (REST_SearchDisplayed)
712 2   {
713 2     /* If the user is currently searching, stop the search */
714 2     REST_SearchCancel ();
715 2
716 2     /* Stop any alarms that were set */
717 2     EVENT_StopCBackAlarm ((ClientPtr)GREST_Handle);
718 2
719 2     REST_SearchDisplayed = BOOL_FALSE;
720 2
721 2     /* Remove the window, but no need to tell us about it */
722 2     WIN_SelfTerminate ((WinPtr)REST_SearchWindow);
723 2
724 2     /* Free the current found list */
725 2     thisObject = firstFoundObject;
726 2     while (thisObject != NULL)
727 3     {
728 3       nextObject = thisObject->next;
729 3       REST_SearchDisposeInfo (thisObject);
730 3       thisObject = nextObject;
731 2     }
732 2     firstFoundObject = NULL;
733 2     lastFoundObject = NULL;
734 2
735 2     REST_SearchWindow = NULL;
736 2
737 2     returnStatus = BOOL_TRUE;
738 1   }
739 1   else
740 2   {
741 2     returnStatus = BOOL_FALSE;
742 1   }
```



```
743 1   /* Return whether or now we actually had to remove the window */
744 1   return (returnStatus);
745   }
```

```
/*****************************************************************
 * Function Name:   REST_SearchWindowIsDisplayed ()
 *
 * Description:
 *   This routine will determin if the search window is currently
 *   displayed.
 *
 * Parameters:
 *   None
 *
 * Success Outputs and Side Effects:
 *   None.
 *
 * Returns:
 *   BOOL_TRUE  - If the window is displayed
 *   BOOL_FALSE - otherwise.
 *
 *****************************************************************/

BoolEnum REST_SearchWindowIsDisplayed (void)
{
  return (REST_SearchDisplayed);
}
```



```
/*****************************************************************
 * Function Name:   REST_SearchGetTime ()
 *
 * Description:
 *   This routine will query the user for a backup date/time and
 *   the date edit widget with the new date if selected.
 *
 * Parameters:
 *   timeTEd (I) - The widget to set the time in.
 *
 * Success Outputs and Side Effects:
 *   None.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_SearchGetTime (TEdPtr  timeTEd)
{
  Char          dateString[SMALL_STRING_LENGTH];  /* New date string */
  GREST_Object currentWI;                         /* Current work item */
  time_t        currentTime;                      /* Current backup time */
  time_t        newTime = 0;                      /* New backup time */

  /* Do nothing if we're searching, or a restore is in progress */
  if (REST_SearchInProgress() || REST_RestoreInProgress())
    return;

  /* Get the currently selected time */
  currentTime = (time_t)WGT_GetClientRes ((WgtPtr)timeTEd);

  /* Get the work item */
  currentWI = REST_GetCurrentWorkItem ();
  if (currentWI != NULL)
  {

    /* Query the user for the new start time */
    newTime = REST_GetUserSelectedTime (currentWI,
                                        currentTime,
                                        (WinPtr)REST_SearchWindow);
  }

  /* If the user selected a time */
  if (newTime != 0)
  {
    /* Get the new date string */
    strftime (dateString,
              SMALL_STRING_LENGTH,
              "%b %d %H:%M",
              localtime (&newTime));

    TED_SetStr ((TEdPtr)timeTEd, dateString);
    WGT_SetClientRes ((WgtPtr)timeTEd, (ClientPtr)newTime);
  }
}
```

```
 828  /*****************************************************************
 829   * REST_GetSearchListColumnValues
 830   *
 831   * Description:
 832   *   This routine will return the drawables for the given information
 833   *   for the given search results list box column.
 834   *
 835   * Parameters:
 836   *   lbox          (I) - The list box to draw to (Not Used)
 837   *   clientData    (I) - The data in the cell. (
                                        a private data structure)
 838   *   row           (I) - The row of the cell
 839   *   column        (I) - The column of the cell
 840   *   text          (O) - The text to draw in the cell.
 841   *   justification (O) - The justification of the text to be drawn.
 842   *   icon1         (O) - The first icon to draw in the cell.
 843   *   icon2         (O) - The second icon to draw in the cell.
 844   *   overlayIcon1  (O) - The first overlayed icon to draw in the cell.
 845   *   overlayIcon2  (
                        O) - The second overlayed icon to draw in the cell.
 846   *
 847   * Returns:
 848   *   BOOL_TRUE  - if the cell should be drawn
 849   *   BOOL_FALSE - otherwise
 850   *
 851   *****************************************************************/

 853  BoolEnum REST_GetSearchListColumnValues (LBoxPtr   lbox,
 854                                           ClientPtr clientData,
 855                                           Int16     row,
 856                                           Int16     column,
 857                                           Str       text,
 858                                           UInt16    *justification,
 859                                           IconPtr   *icon1,
 860                                           IconPtr   *icon2,
 861                                           IconPtr   *overlayIcon1,
 862                                           IconPtr   *overlayIcon2)
 863  1 {
 864  1   REST_FoundObjectPtr info;
                                  /* The real data from the client data */
 865  1   u_hyper              size;    /* Size of the item */
 866  1   time_t               theTime; /* Date for the item */
 867  1   time_t               now;     /* The current time */
 868  1   BoolEnum             redraw;
                                  /* Flag whether or not to draw this cell */

 870  1   /* Get the real pointer */
 871  1   if ((column == 1) || (clientData != NULL))
 872  2   {
 873  2     info = (REST_FoundObjectPtr) clientData;
 874  1   }
 875  1   else
 876  2   {
 877  2     LBOX_GoCellRow (lbox, row);
 878  2     info = (REST_FoundObjectPtr) LBOX_CurGetClientData (lbox);
 879  2     LBOX_GoCellRow (lbox, column, row);
 880  1   }

 882  1   /* Start with the default justification */
 883  1   *justification = DRAW_JUSTLEFT | DRAW_JUSTVCENTER;

 885  1   /* Never a second icon or overlay icon */
 886  1   *icon2 = NULL;
 887  1   *overlayIcon2 = NULL;
```



```
 889  1   /* Initialize everything to blank */
 890  1   *icon1 = NULL;
 891  1   *overlayIcon1 = NULL;
 892  1   STR_Cpy (text, "");

 894  1   /* Determine each value based on the given info and the column */
 895  1   if ((info != NULL) && (column <= NUMBER_FOUND_COLUMNS))
 896  2   {
 897  2     /* Draw this cell */
 898  2     redraw = BOOL_TRUE;

 900  2     if (column == BACKUP_TIME_COLUMN)
 901  3     {
 902  3       /* get the backup time string */
 903  3       strftime (text,
 904  3                 GMAX_CELL_STRING_LENGTH,
 905  3                 "[%c]",
 906  3                 localtime(&info->backupTime));
 907  2     }

 909  2     else if (column == MARK_COLUMN)
 910  3     {
 911  3       *icon1 = searchCheckBoxIcon;

 913  3       /* Get the overlay icon */
 914  3       if (GREST_IsObjectMarkedForTime (
                                  GREST_Handle, info->object, info->backupTime))
 915  4       {
 916  4         if (LBOX_IsCurSelected (lbox))
 917  4           *overlayIcon1 = searchSelCheckIcon;
 918  4         else
 919  4           *overlayIcon1 = searchCheckIcon;
 920  3       }
 921  3       else
 922  4       {
 923  4         *overlayIcon1 = NULL;
 924  3       }
 925  2     }
 926  2     else if (column == ICON_COLUMN)
 927  3     {
 928  3       /* Get the type icon */
 929  3       if (EDMRST_IsObjectContainer (GREST_Handle, info->object))
 930  4       {
 931  4         *icon1 = searchDirIcon;
 932  3       }
 933  3       else
 934  4       {
 935  4         *icon1 = searchFileIcon;
 936  3       }

 938  3       *overlayIcon1 = REST_GetStatusIcon (EDMRST_GetObjectStatus(
                                                          GREST_Handle,
 939  3                                                       info->object));

 940  2     }

 942  2     /* Column 3 is the Name column */
 943  2     else if (column == NAME_COLUMN)
 944  3     {
 945  3       /* get the string for the full name */
 946  3       STR_Cpy (text, EDMRST_GetObjectFullName (
                                                  GREST_Handle, info->object));
 947  2     }

 949  2     /* Column 4 is the Name column */
 950  2     else if (column == PERMISSIONS_COLUMN)
```

```
951  3     {
952  3       /* get the string for the permissions */
953  3       STR_Cpy (text, GREST_GetPermissionsString (
                                                  GREST_Handle, info->object));
954  2     }

956  2     else if (column == OWNER_COLUMN)
957  2     {
958  3       /* get the string for the owner */
959  3       STR_Cpy (text, EDMRST_GetObjectOwnerString (
                                                  GREST_Handle, info->object));
960  2     }

962  2     else if (column == GROUP_COLUMN)
963  2     {
964  3       /* get the string for the group */
965  3       STR_Cpy (text, EDMRST_GetObjectGroupString (
                                                  GREST_Handle, info->object));
966  2     }


968  2     else if (column == SIZE_COLUMN)
969  2     {
970  3       /* get the string for the size */
971  3       size = EDMRST_GetObjectSize (GREST_Handle, info->object);
972  3       REST_SPrintHyper (text, size);
973  3       *justification = DRAW_JUSTRIGHT | DRAW_JUSTVCENTER;
974  2     }

976  2     else if (column == DATE_COLUMN)
977  2     {
978  3       /* get the string for the date */
979  3       theTime = EDMRST_GetObjectModDate (GREST_Handle, info->object);
980  3       strftime (text, GMAX_CELL_STRING_LENGTH, "%b %d", localtime (
                                                        &theTime));

981  2     }

983  2     else if (column == TIME_COLUMN)
984  2     {
985  3       /* get the string for the time */
986  3       theTime = EDMRST_GetObjectModDate (GREST_Handle, info->object);
987  3       now = time((time_t *)NULL);

989  3       /* If the object is over a year old, display the year */
990  3       if ((now - theTime) > SECONDS_PER_YEAR)
991  4       {
992  4         strftime (text, GMAX_CELL_STRING_LENGTH, "%y", localtime (
                                                        &theTime));

993  4       }
994  3       /* Otherwise display the time */
995  3       else
996  4       {
997  4         strftime (text, GMAX_CELL_STRING_LENGTH, "%H:%M", localtime (
                                                        &theTime));

998  4       }
999  3     }
1000 2   }
1001 1   else
1002 2   {
1003 2     /* Don't draw this cell */
1004 2     redraw = BOOL_TRUE;
1005 1   }


1008 1   /* Return whether or not to redraw */
1009 1   return (redraw);
1010 }
```



```
1012 /*****************************************************************
1013  * REST_SearchInProgress
1014  *
1015  * Description:
1016  *   This routine returns whether or not a search is currently in
                                                              progress.
1017  *
1018  * Parameters:
1019  *   None.
1020  *
1021  * Returns:
1022  *   BOOL_TRUE  - If a search has been started (
                                                  and is not yet finished)
1023  *   BOOL_FALSE - Otherwise.
1024  *
1025  *****************************************************************/

1027 BoolEnum  REST_SearchInProgress (void)
1028 1 {
1029 1   return (searchInProgress);
1030 }
```

```
1032  /*****************************************************************
1033   * REST_SearchFindTimeout                                        *
1034   *                                                               *
1035   * Description:                                                  *
1036   *    This routine checks for the next set of find results       *
1037   *                                                               *
1038   * Parameters:                                                   *
1039   *    clientData - Not used                                      *
1040   *                                                               *
1041   * Returns:                                                      *
1042   *    None.                                                      *
1043   *                                                               *
1044   *****************************************************************/
1045
1046  static void REST_SearchFindTimeout (ClientPtr ClientData)
1047  1 {
1048  1   long          numEntries;                    /* Number found */
1049  1   Char          statusString[MEDIUM_STRING_LENGTH];
                                                     /* Search status string */
1050  1   GREST_Object  foundObjects[MAX_FOUND_OBJECTS];
                                                     /* Array of objects found */
1051  1   time_t        times[MAX_FOUND_OBJECTS];
                                                     /* Backup times for objects*/
1052  1   Int           i;                             /* Loop index */
1053  1   eerrno_ty     eerrno;
                                                     /* Error code returned */
1054  1   BackupStatus  objectStatus;
                                                     /* Current object's status */
1055  1   BoolEnum      isCancelled;                   /* Flag to if cancelled */
1056  1   BoolEnum      displayError = BOOL_FALSE;
                                                     /* Flag to display error */
1057  1   BoolEnum      allowAll;
                                                     /* Flag if all status is OK*/
1058  1   BoolEnum      allowGood;
                                                     /* Flag if only Good is OK */
1059  1   BoolEnum      allowBad;
                                                     /* Flag if only bad is OK */
1060
1061  1   /* Allocate space for the next array of objects */
1062  1   if (EDMRST_AllocRestorableObjects (GREST_Handle,
1063  1                                      foundObjects,
1064  1                                      MAX_FOUND_OBJECTS) == E_SUCCESS)
1065  2   {
1066  2     /* Initialize this pass */
1067  2     numEntries = 0;
1068
1069  2     isCancelled = REST_CheckForCancel();
1070
1071  2     if ((eerrno = EDMRST_GetFindResults (GREST_Handle,
1072  2                                          isCancelled,
1073  2                                          MAX_FOUND_OBJECTS,
1074  2                                          foundObjects,
1075  2                                          times,
1076  2                                          &numEntries,
1077  2                                          &REST_SearchCookie)) ==
                                                          E_SUCCESS)
1078  3     {
1079  3       /* Get the current status flags */
1080  3       allowAll  =TBUT_GetSelected((
                                  TButPtr)REST_SearchWindow->AllStatusToggle);
1081  3       allowGood =TBUT_GetSelected((
                                  TButPtr)REST_SearchWindow->GoodStatusToggle);
1082  3       allowBad  =TBUT_GetSelected((
                                  TButPtr)REST_SearchWindow->BadStatusToggle);
```



```
1084  3       /* Update the number of entries found so far */
1085  3       REST_CurrentEntries += numEntries;

1087  1       /* Add each found object */
1088  1       for (i=0; i<numEntries; i++)
1089  4       {
1090  4         /* Make sure it passes our internal test for status (
                                                          not in API) */
1091  4         objectStatus =  EDMRST_GetObjectStatus (
                                                GREST_Handle, foundObjects[i]);

1092  4         if (allowAll ||
1093  4             (allowGood && (objectStatus == Backup_Good)) ||
1094  4             (allowBad && (objectStatus == Backup_Bad)))
1095  5         {
1096  5           /* Add the object to the list of found objects */
1097  5           REST_AddFoundItem (foundObjects[i], times[i]);
1098  4         }
1099  4         else
1100  5         {
1101  5           /* This one don't pass, decrement the number found */
1102  5           REST_CurrentEntries --;

1104  5           /* Free this object */
1105  5           EDMRST_FreeRestorableObjects (
                                                GREST_Handle, &foundObjects[i], 1);

1106  4         }
1107  3       }
1108  2     }
1109  2     else if (eerrno == EP_RB_RECOVER_FIND_INTERRUPTED)
1110  3     {
1111  3       /* The search was cancelled */
1112  3       GALERT_DisplayError ((WinPtr)REST_SearchWindow,
1113  3                            REST_GetErrorString (
                                                REST_CANCEL_SEARCH_TITLE),
                                    GICON_GetError(),
1115  3                            REST_GetErrorString (
                                                REST_CANCEL_SEARCH_MESSAGE));


1117  3       /* We're done */
1118  3       REST_SearchCookie = DONE_COOKIE;
1119  2     }
1120  2     else if (eerrno != EP_RB_RECOVER_RPC_INCOMPLETE)
1121  3     {
1122  3       /* Flag that we want to display an error message */
1123  3       displayError = BOOL_TRUE;

1125  3       /* We're done */
1126  3       REST_SearchCookie = DONE_COOKIE;
1127  2     }

1129  2     /* Free up the left overs */
1130  2     if (numEntries < MAX_FOUND_OBJECTS)
1131  3     {
1132  3       EDMRST_FreeRestorableObjects (GREST_Handle,
1133  3                                     &foundObjects[numEntries],
1134  3                                     MAX_FOUND_OBJECTS - numEntries);
1135  2     }
1136  1   }
1137  1   else
1138  2   {
1139  2     /* Error allocating restorable objects, boogie on out */
1140  2     REST_SearchCookie = DONE_COOKIE;
1141  1   }

1143  1   if (REST_SearchCookie == DONE_COOKIE)
```

```
1144 4    {
1145 2      /* Flag that the search is done */
1146 2      searchInProgress = BOOL_FALSE;

1148 2    /* Make sure the window is still active */
1149 2    if (REST_SearchWindow != NULL)
1150 3    {
1151 3      /*
1152 3       * Update the search result status
1153 3       */

1155 3      /* Update the status text */
1156 3      STR_Sprintf (statusString, "%ld", REST_CurrentEntries);
1157 3      TSD_SetStr ((
                    TEdPtr)REST_SearchWindow->ItemsFoundText, statusString);

1159 3      /* Fill the listbox */
1160 3      GUTIL_LBOX_FillVirtual (
                          REST_SearchWindow->FoundBox, REST_CurrentEntries);

1162 3      /* Update the search results action buttons */
1163 3      REST_SearchUpdateButtons ();

1165 3      /* Remove the working dialog */
1166 3      REST_SearchCancel ();

1168 3      /* If there was an error, display the error message */
1169 3      if (displayError)
1170 4      {
1171 4        /* Process events to remove the in progress dialog */
1172 4        EVENT_ProcessPending ();

1174 4        /* Display the current error message */
1175 4        REST_DisplayErrnoMessage ((WinPtr)REST_SearchWindow,
1176 4                                  REST_GetErrorString (
                                                REST_SEARCH_FAIL_TITTLE),
1177 4                                  NULL,
1178 4                                  eerrno);
1179 3      }
1180 2    }
1181 1    }
1182 1    else
1183 1    {
1184 2      if (eerrno == EP_KB_RECOVER_RPC_INCOMPLETE)
1185 3      {
1186 3        EVENT_StartCBackAlarm (REST_SearchFindTimeout,
1187 3                               clientData,
1188 3                               RESTORE_START_FIND_RESULTS_DELAY);
1189 2      }
1190 2      else
1191 3      {
1192 3        EVENT_StartCBackAlarm (REST_SearchFindTimeout,
1193 3                               clientData,
1194 3                               RESTORE_CONT_FIND_RESULTS_DELAY);
1195 2      }
1196 1    }
1197  }
```



```
1199  /*************************************************************
1200   * Function Name:   REST_SearchStartSearch ()
1201   *
1202   * Description:
1203   *   This routine will start the search based on current criteria
1204   *   found in the widgets.
1205   *
1206   * Parameters:
1207   *   None.
1208   *
1209   * Success Outputs and Side Effects:
1210   *   The search results widgets will be updated.
1211   *
1212   * Returns:
1213   *   None.
1214   *
1215   *************************************************************/

1217  void REST_SearchStartSearch (void)
1218 1 {
1219 1   EBREC_SearchCriteriaRec searchRec;
                                          /* Criteria for search */
1220 1   Char            sizeString[SMALL_STRING_LENGTH];
                                          /* Size criteria from TEd */
1221 1   eerrno_ty       eerrno;
                                          /* Error code returned */

1223 1   /* Do nothing if we're searching, or a restore is in progress */
1224 1   if (REST_SearchInProgress() || REST_RestoreInProgress())
1225 1     return;

1227 1   /* Get the search directory */
1228 1   TED_QueryStr ((TEdPtr)REST_SearchWindow->DirectoryText,
1229 1                 searchRec.startDirectory,
1230 1                 MAX_STRING_LENGTH);

1232 1   /* Standardize pathname: convert NT notation to UNIX */
1233 1   REST_StandardizePath(searchRec.startDirectory);

1235 1   if (STR_Cmp (searchRec.startDirectory, "") == CMP_EQUAL)
1236 1     STR_Cpy (searchRec.startDirectory, "/");

1238 1   /* Get whether or not to descend into sub directories */
1239 1   searchRec.descendDirectory = TBUT_GetSelected ((
                                        TButPtr)REST_SearchWindow->DescendToggle);

1241 1   /* Get the search string */
1242 1   TED_QueryStr ((
              TEdPtr)REST_SearchWindow->StringText, searchRec.searchString,
                                                    MAX_FILENAME_LENGTH);

1244 1   /* Standardize pathname: convert NT notation to UNIX */
1245 1   REST_StandardizePath(searchRec.searchString);

1247 1   if (STR_Cmp (searchRec.searchString, "") == CMP_EQUAL)
1248 2   {
1249 2     /* Default to everything, (
                                    exluding nothing = include everything) */
1250 2     STR_Cpy (searchRec.searchString, "*");
1251 2     searchRec.excludeString = BOOL_FALSE;
1252 1   }
1253 1   else
1254 2   {
1255 2     /* Get whether or not to exclude the search string */
1256 2     searchRec.excludeString = TBUT_GetSelected ((
```

```
                                TButPtr)REST_SearchWindow->ExcludeRadio);
1257 1  }
1258 1
1259 1  /* Get the type of objects to search for */
1260 1  if (TBUT_GetSelected((TButPtr)REST_SearchWindow->AllTypesToggle))
1261 1    searchRec.typeOfFile = All_File_Types;
1262 1  else if (TBUT_GetSelected((
                                TButPtr)REST_SearchWindow->DirectoriesToggle))
1263 1    searchRec.typeOfFile = Directories_Only;
1264 1  else if (TBUT_GetSelected((TButPtr)REST_SearchWindow->FilesToggle))
1265 1    searchRec.typeOfFile = Files_Only;
1266 1  else
1267 1    searchRec.typeOfFile = Others_Only;
1268 1
1269 1  /* Get the owner to limit the search to */
1270 1  TED_QueryStr ((
                        TEdPtr)REST_SearchWindow->OwnerText, searchRec.owner,
                                                MAX_FILENAME_LENGTH);


1272 1  /* Get whether or not to exclude the owner */
1273 1  searchRec.excludeOwner = TBUT_GetSelected ((
                                TButPtr)REST_SearchWindow->ExcludeOwnerToggle);

1275 1  /* Get the group to limit the search to */
1276 1  TED_QueryStr ((
                        TEdPtr)REST_SearchWindow->GroupText, searchRec.group,
                                                MAX_FILENAME_LENGTH);

1278 1  /* Get whether or not to exclude the group */
1279 2  searchRec.excludeGroup = TBUT_GetSelected ((
                                TButPtr)REST_SearchWindow->ExcludeGroupToggle);

1281 1  /* Get the size string */
1282 1  TED_QueryStr ((
TEdPtr)REST_SearchWindow->SizeText, sizeString, SMALL_STRING_LENGTH);
1283 1  if ((sizeString != NULL) && (STR_Len(sizeString) > 0))
1284 2  {
1285 2    /* Convert the string to a hyper */
1286 2    REST_ScanHyper (sizeString, &searchRec.sizeInBytes);

1288 2    /* Get the size criteria */
1289 2    if (TBUT_GetSelected((
                                TButPtr)REST_SearchWindow->GreaterThanToggle))
1290 2      searchRec.sizeMatch = EBREC_GreaterThan;
1291 2    else if (TBUT_GetSelected((
                                TButPtr)REST_SearchWindow->LessThanToggle))
1292 2      searchRec.sizeMatch = EBREC_LessThan;
1293 2    else
1294 2      searchRec.sizeMatch = EBREC_Equal;
1295 1  }
1296 1  else
1297 2  {
1298 2    /* Search for all sizes */
1299 2    searchRec.sizeMatch = EBREC_AllSizes;
1300 1  }

1302 1  /* Get the start backup date/time */
1303 1  searchRec.startTime = (time_t)WGT_GetClientRes ((
                                WgtPtr)REST_SearchWindow->StartDateOutput);

1305 1  /* Get the end backup date/time */
1306 1  searchRec.endTime = (time_t)WGT_GetClientRes ((
                                WgtPtr)REST_SearchWindow->EndDateOutput);

1308 1  /* Flush the event queue */
1309 1  EVENT_DiscardUserEvents ();
```



```
1310 1  EVENT_ProcessPending ();

1312 1  /* Initialize the number of entries found so far */
1313 1  REST_CurrentEntries = 0;

1315 1  /* Flag that the search is in progress */
1316 1  searchInProgress = BOOL_TRUE;

1318 1  /* Clear the previous entries */
1319 1  REST_ClearFoundBox ();

1321 1  /* Find the next group of objects */
1322 1  if ((errno = EDMRST_FindRestorableObjects (
                                        GREST_Handle, &searchRec)) == E_SUCCESS)
1323 2  {
1324 2    REST_SearchCookie = INIT_COOKIE;
1325 3    EVENT_StartBackAlarm (REST_SearchFindTimeout,
1326 2                          (ClientPtr)GREST_Handle,
1327 2                          RESTORE_START_FIND_RESULTS_DELAY);
1328 1  }
1329 1  else
1330 2  {
1331 2    /* Display the current error message */
1332 2    REST_DisplayErrnoMessage ((WinPtr)REST_SearchWindow,
1333 2                              REST_GetErrorString (
                                                REST_SEARCH_FAIL_TITLE),
1334 2                              NULL,
1335 2                              errno);
1336 1  }
1337 }
```

```
1339  /*****************************************************************
1340   * Function Name:   REST_SearchToggleMark ()
1341   *
1342   * Description:
1343   *   This routine will mark/unmark the give item.
1344   *
1345   * Parameters:
1346   *   object (I) - The object to toggle the marking of
1347   *
1348   * Success Outputs and Side Effects:
1349   *   The object will be marked or unmarked for restore.
1350   *
1351   * Returns:
1352   *   None.
1353   *
1354   *****************************************************************/

1356  void REST_SearchToggleMark (ClientPtr object)
1357  1 {
1358  1   REST_FoundObjectPtr   info;
                                          /* Next selected object */
1359  1   BoolEnum              marked = BOOL_FALSE;
                                        /* Flag if any marks worked */
1360  1   long                  numberMarked = 0;    /* Number of marks */
1361  1   long                  numberBad = 0;
                                          /* Number of bad marks */
1362  1   u_hyper               totalSize;           /* Total size marked */

1364  1   /* Do nothing if we're searching, or a restore is in progress */
1365  1   if (REST_SearchInProgress() || REST_RestoreInProgress())
1366  1     return;

1368  1   info = (REST_FoundObjectPtr)object;
1369  1   if (info != NULL)
1370  2   {
1371  2     /* Determine if this object is markable */
1372  2     if (GREST_IsObjectMarkableForTime (GREST_Handle,
1373  3                                        info->object,
1374  3                                        info->backupTime) &&
1375  4         ((EDMRST_GetObjectStatus (GREST_Handle,
1376  4                                   info->object) != Backup_Bad) ||
1377  3          REST_MarkBadFiles))
1378  3     {
1379  3       if (!GREST_IsObjectMarkedForTime (
                          GREST_Handle, info->object, info->backupTime))
1380  4       {
1381  4         marked = REST_MarkRestorableObject (info->object,
1382  4                                             info->backupTime,
1383  4                                             &numberMarked,
1384  4                                             &numberBad);
1385  3       }
1386  3       /* Otherwise, unmark it for restore */
1387  3       else
1388  4       {
1389  4         marked = REST_UnmarkRestorableObject (info->object,
1390  4                                               info->backupTime,
1391  4                                               &numberMarked,
1392  4                                               &numberBad);
1393  3       }

1395  3       /* Update the mark flags for all objects */
1396  3       REST_UpdateObjectMarks (currentWorkItemInfo);
1397  2     }

1399  2     if (marked)
```



```
1400  3     {
1401  3       /* Redrew the list boxes, to update the current marks */
1402  3       SAREA_RedrawParts ((SAreaPtr)REST_SearchWindow->PoundBox);
1403  3       SAREA_RedrawParts ((SAreaPtr)REST_RestoreWin->BackupListBox);
1404  3       SAREA_RedrawParts ((SAreaPtr)REST_RestoreWin->SelectedListBox);

1406  3       /* Update the mark information */
1407  3       EDMRST_GetMarkedTotalSize (GREST_Handle, &totalSize);
1408  3       REST_UpdateMarkedInfo (numberMarked,
1409  3                              totalSize,
1410  3                              numberBad);

1412  3       /* Update the search results action buttons */
1413  3       REST_SearchUpdateButtons ();
1414  2     }
1415  1   }
1416  }
```

```
1418  /*****************************************************************
1419   * Function Name:   REST_SearchSetMark ()
1420   *
1421   * Description:
1422   *   This routine will mark/unmark all selected items in the search
1423   *                                                          results
1424   *   list box.
1425   *
1426   * Parameters:
1427   *   setMark (I) - Flag if the items should be marked or unmarked.
1428   *
1429   * Success Outputs and Side Effects:
1430   *   The selected items will be marked or unmarked for restore.
1431   *
1432   * Returns:
1433   *   None.
1434   *
1435   *****************************************************************/
1436  void REST_SearchSetMark (BoolEnum setMark)
1437  1 {
1438  1   REST_FoundObjectPtr    info;
                                                  /* Next selected object */
1439  1   long                   numberMarked;     /* Number of marks */
1440  1   long                   numberBad;        /* Number of bad marks */
1441  1   BoolEnum               thisMark;
                                                  /* Flag if this mark worked */
1442  1   BoolEnum               marked = BOOL_FALSE;
                                                  /* Flag if any marks worked */
1443  1   u_hyper                totalSize;        /* Total marked size */

1445  1   /* Do nothing if we're searching, or a restore is in progress */
1446  1   if (REST_SearchInProgress() || REST_RestoreInProgress())
1447  1     return;

1449  1   /* If any rows are selected */
1450  1   if (LBOX_GoRowSelFirst (REST_SearchWindow->FoundBox) == BOOL_TRUE)
1451  2   {

1453  2     /* Go to the first row */
1454  2     LBOX_GoHome (REST_SearchWindow->FoundBox);

1456  2     /* Loop through all rows that have data */
1457  2     while ((info = (REST_FoundObjectPtr)LBOX_CurGetClientData(
                                  REST_SearchWindow->FoundBox)) != NULL)
1458  3     {
1459  3       /* If this row is selected, set the mark appropriately */
1460  3       if (LBOX_IsCurSelected (REST_SearchWindow->FoundBox))
1461  4       {
1462  4         /* If this is a mark operation, then mark it for restore */
1463  4         if (setMark)
1464  5         {
1465  5           if (!GREST_IsObjectMarkedForTime (
                            GREST_Handle, info->object, info->backupTime))
1466  6           {
1467  6             thisMark = REST_MarkRestorableObject (info->object,
1468  6                                                   info->backupTime,
1469  6                                                   &numberMarked,
1470  6                                                   &numberBad);
1471  5           }
1472  5         }
1473  4         /* Otherwise, unmark it for restore */
1474  4         else
1475  5         {
1476  5           if (GREST_IsObjectMarkedForTime (
```



```
                            GREST_Handle, info->object, info->backupTime))
1477  6           {
1478  6             thisMark = REST_UnmarkRestorableObject (info->object,
1479  6                                                     info->backupTime,
1480  6                                                     &numberMarked,
1481  6                                                     &numberBad);
1482  5           }
1483  4         }

1485  4         /* Set the flag if anything has been marked yet */
1486  4         marked = (marked || thisMark);
1487  3       }

1489  3       /* Go to the next row */
1490  3       LBOX_GoDown (REST_SearchWindow->FoundBox);
1491  2     }
1492  1   }

1494  1   /* Update the mark flags for all objects */
1495  1   REST_UpdateObjectMarks (currentWorkItemInfo);

1497  1   /* If any mark or unmark was succesful, update the display */
1498  1   if (marked)
1499  2   {
1500  2     /* Redraw the list boxes, to update the current marks */
1501  2     SAREA_RedrawParts ((SAreaPtr)REST_SearchWindow->FoundBox);
1502  2     SAREA_RedrawParts ((SAreaPtr)REST_RestoreWin->BackupListBox);
1503  2     SAREA_RedrawParts ((SAreaPtr)REST_RestoreWin->SelectedListBox);

1505  2     /* Update the mark information */
1506  2     EDMRST_GetMarkedTotalSize (GREST_Handle, &totalSize);
1507  2     REST_UpdateMarkedInfo (numberMarked,
1508  2                            totalSize,
1509  2                            numberBad);
1510  1   }

1512  1   /* Update the search results action buttons */
1513  1   REST_SearchUpdateButtons ();
1514  }
```

```
1516  /*****************************************************************
1517   * Function Name:   REST_SearchSetView ()
1518   *
1519   * Description:
1520   *   This routine will set the current restore window's view to
1521   *   the currently selected item in the search results list box.
1522   *   This routine assumes only one item is selected, if multiples
1523   *   are selected, only the first selected item will be acted upon.
1524   *
1525   * Parameters:
1526   *   None.
1527   *
1528   * Success Outputs and Side Effects:
1529   *   The selected item will be in the restore view.
1530   *
1531   * Returns:
1532   *   None.
1533   *
1534   *****************************************************************/

1536  void REST_SearchSetView (void)
1537  1 {
1538  1   REST_FoundObjectPtr info;
                                        /* Info for the selected row */
1539  1   BoolEnum            found = BOOL_FALSE;
                                        /* Flag if we found the row */
1540  1   Str                 fullName;
                                        /* Full name of the object */

1542  1   /* Do nothing if we're searching, or a restore is in progress */
1543  1   if (REST_SearchInProgress() || REST_RestoreInProgress())
1544  1     return;

1546  1   /* If anything is selected */
1547  1   if (LBOX_GoRowSelFirst (REST_SearchWindow->FoundBox) == BOOL_TRUE)
1548  2   {
1549  2     /* Go to the first row */
1550  2     LBOX_GoHome (REST_SearchWindow->FoundBox);

1552  3     /* Loop through all rows that have data */
1553  3     info = (REST_FoundObjectPtr)LBOX_CurGetClientData(
                                          REST_SearchWindow->FoundBox);
1554  3     while ((!found) && (info != NULL))
1555  4     {
1556  4       /* If this row is selected, set the view */
1557  4       if (LBOX_IsCurSelected (REST_SearchWindow->FoundBox))
1558  5       {
1559  5         /* Set the backup view to this object */
1560  5         fullName = esl_strdup (EDMREST_GetObjectFullName (
                                          GREST_Handle, info->object));
1563  5         REST_StripDirectoryChars (fullName);

1565  5         REST_SetBackupView (info->backupTime, fullName);

1567  5         GUTIL_Free (fullName);

1568  5         /* Flag that we found it */
1569  5         found = BOOL_TRUE;
1570  4       }
1571  4       /* Otherwise, go to the next row */
1572  5       else
1573  5       {
1574  6         LBOX_GoDown (REST_SearchWindow->FoundBox);
1575  6         info = (REST_FoundObjectPtr)LBOX_CurGetClientData(
                                          REST_SearchWindow->FoundBox);
1575  3       }
1576  2     }
1577  1   }
1578  }
```

```
1580  /*****************************************************************
1581   * Function Name:    REST_ClearFoundBox ()
1582   *
1583   * Description:
1584   *    This routine will clear all items from the search results list
                                                                    box.
1585   *
1586   * Parameters:
1587   *    None.
1588   *
1589   * Success Outputs and Side Effects:
1590   *    The search results list box will be empty.
1591   *
1592   * Returns:
1593   *    None.
1594   *
1595   *****************************************************************/

1597  void REST_ClearFoundBox (void)
1598  1 {
1599  1   REST_FoundObjectPtr thisObject;
1600  1   REST_FoundObjectPtr nextObject;

1602  1   /* Free the current found list */
1603  1   thisObject = firstFoundObject;
1604  1   while (thisObject != NULL)
1605  2   {
1606  2     nextObject = thisObject->next;
1607  2     REST_SearchDisposeInfo (thisObject);
1608  2     thisObject = nextObject;
1609  1   }
1610  1   firstFoundObject = NULL;
1611  1   lastFoundObject = NULL;

1613  1   /* Set everything to zero and blank */
1614  1   TED_SetStr ((TEdPtr)REST_SearchWindow->ItemsFoundText, "");

1616  1   /* Reset the list box and start at the beginning */
1617  1   LBOX_AllUnselect (REST_SearchWindow->FoundBox);
1618  1   LBOX_Reset (REST_SearchWindow->FoundBox);
1619  1   LBOX_GoHome (REST_SearchWindow->FoundBox);

1621  1   GUTIL_LBOX_FillVirtual (REST_SearchWindow->FoundBox, 0);

1623  1   /* Update the search results action buttons */
1624  1   REST_SearchUpdateButtons ();
1625    }
```



```
1627  /*****************************************************************
1628   * Function Name:    REST_SearchInitialize ()
1629   *
1630   * Description:
1631   *    This routine will initialize the use of the search window
                                                                functions.
1632   *
1633   * Parameters:
1634   *    None.
1635   *
1636   * Success Outputs and Side Effects:
1637   *    None.
1638   *
1639   * Returns:
1640   *    None.
1641   *
1642   *****************************************************************/

1644  void REST_SearchInitialize (void)
1645  1 {
1646  1   static BoolEnum initialized = BOOL_FALSE;
                                        /* Flag if we have init'd yet */

1648  1   /* If we haven't initialized already, initialize */
1649  1   if (!initialized)
1650  2   {

1652  2     /* Get the icons */
1653  2     searchDirIcon   = GICON_GetIconBySize (I_DIRCLOSED, ICON_SMALL);
1654  2     searchFileIcon  = GICON_GetIconBySize (I_FILE, ICON_SMALL);
1655  2     searchCheckBoxIcon = GICON_GetIconBySize (I_CHECKBOX, ICON_SMALL);
1656  2     searchCheckIcon = GICON_GetIconBySize (I_CHECK, ICON_SMALL);
1657  2     searchSelCheckIcon = GICON_GetIconBySize (
                                                  I_SELECTED_CHECK, ICON_SMALL);
1658  2     searchOffsiteIcon     = GICON_GetIconBySize (I_OFFSITE, ICON_SMALL);
1659  2     searchBadIcon = GICON_GetIcon (I_BADBUOBJECT);

1661  2     /* Flag that we have already initialized */
1662  2     initialized = BOOL_TRUE;
1663  1   }
1664    }
```

```
1  /*****************************************************************
2   * restUtils.c
3   *
4   *
5   * Copyright 1996 by Epoch Systems, Inc.
6   *
7   *
8   * Mission Statement:
9   *    This file contains the utility functions for the EDM Restore
                                                              window.
10  *
11  * Required Includes:
12  *    None
13  *
14  * Compile-Time Options:
15  *    N/A
16  *
17  *
18  * RCS Information:
19  *       $RCSfile$
20  *       $Revision$
21  *       $Date$
22  *****************************************************************/
23
24  #define  ERR_LIB  RESTORE
25
26  #include <esl/c_portable.h>
27  #include <esl/ep_xopen.h>
28
29  #include <stdlib.h>
30
31  #include <libgen.h>
32  #include <time.h>
33
34  #include <apppub.h>
35  #include <eventpub.h>
36  #include <rlibpub.h>
37  #include <gwpub.h>
38  #include <respub.h>
39  #include <strlpub.h>
40
41  #include "errno/e_errno.h"
42  #include <util/esl_string.h>
43  #include <restore/restore_api.h>
44  #include "restore/restMgr.h"
45  #define REST_UTIL_INIT
46  #include "restUtils.h"
47  #undef REST_UTIL_INIT
48  #include "restore.h"
49  #include "restoreP.h"
50  #include "restCBMgr.h"
51  #include "restAPIUtils.h"
52  #include "gutil/timeUtils.h"
53  #include "gutil/iconUtils.h"
54  #include "gutil/winUtils.h"
55  #include "gutil/msgUtils.h"
56  #include "gutil/miscUtils.h"
57  #include "gutil/FileMgr.h"
58  #include "gutil/guiDefines.h"
59  #include "gutil/gutUtils.h"
60  #include "gutil/alertMgr.h"
61
62  ERR_EXTERN
63  ERR_INMODULE("restore")
64
65  /*************
```



```
66  * Constants *
67  *************/
68
69  #define REST_MESSAGE_HEADER "RESTORE"
70
71  /*************
72   * Globals   *
73   *************/
74
75  static GUTIL_MsgHandle restMsgList = NULL;
76
77  /*****************************************************************
78   * REST_UtilInitialize
79   *
80   * Description:
81   *    This routine will initialize the routines in the utilities
                                                              portion of
82   *    the restore functionality.
83   *
84   * Parameters:
85   *    None.
86   *
87   * Returns:
88   *    None.
89   *
90   *****************************************************************/
91
92  void REST_UtilInitialize (void)
93  1 {
94  1   /* Initialize the message utilities */
95  1   restMsgList = GUTIL_MsgHandleInit ("restore", "RestoreStrL");
96    }
```

```
/*****************************************************************
 * REST_GetErrorString
 *
 * Description:
 *   This routine will retrieve the error string for the given error
 *   index.
 *
 * Parameters:
 *   errorIndex (I) - Index of the error string to retrieve
 *
 * Returns:
 *   Static pointer to the error string (Do not overwrite!)
 *
 *****************************************************************/

Str REST_GetErrorString (Int       errorIndex)
{
  Char  messageCode[SMALL_STRING_LENGTH];
                                  /* Built code in string format */

  /* Build the message code using the given index */
  STR_Sprintf (messageCode, "%s%d", REST_MESSAGE_HEADER, errorIndex);

  /* Return the string at the index into the error message list */
  return (GUTIL_MsgGetMsg (restMsgList, messageCode));
}
```



```
/*****************************************************************
 * REST_DisplayErrnoMessage
 *
 * Description:
 *   This routine will display the error dialog with the given message
 *   and the text string for the given error code.  If the title given
 *   is NULL, the title "Error" will be used.  If the errorText given
 *   is NULL, only the eerrno error string will be displayed.
 *
 * Parameters:
 *   title     (I) - Title of the error window (may be NULL)
 *   errorText (I) - Text to display (may be NULL)
 *   eerrno    (I) - The error code
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_DisplayErrnoMessage (WinPtr     parent,
                               Str        title,
                               Str        errorText,
                               eerrno_ty eerrno)
{
  Char        outputString[MAX_STRING_LENGTH];  /* String to display */
  Char        tmpTitle[MAX_STRING_LENGTH];      /* Title to display */

  /* Create the error text string to display */
  if (errorText != NULL)
    STR_Sprintf (outputString, "%s\n%s", errorText, e_get_error_text(
                                                          eerrno));
  else
    STR_Cpy (outputString, e_get_error_text(eerrno));

  /* Create the title to display */
  if (title == NULL)
    STR_Cpy (tmpTitle, REST_GetErrorString (REST_ERROR_INDEX));
  else
    STR_Cpy (tmpTitle, title);

  /* Display the error window */
  GALERT_DisplayError (parent,
                       tmpTitle,
                       GICON_GetError(),
                       outputString);
}
```

```
170   /*****************************************************************
171    * REST_SPrintHyper
172    *
173    * Description:
174    *   This routine will fill the given string with a textual
                                                          representation
175    *   of the given hyper.
176    *
177    * Parameters:
178    *   string (I) - The pre-allocated string to fill
179    *   size   (I) - The hyper to represent
180    *
181    * Returns:
182    *   None.
183    *
184    *****************************************************************/

186   void REST_SPrintHyper (Str     string,
187                          u_hyper size)
188 1 {
189 1   /* Print the hyper to the string */
190 1   if (hyper_is_ulong(size))
191 1     STR_Sprintf (string, "%lu", size.low);
192 1   else
193 1     STR_Sprintf (string, "%s", format_u_hyper_for_output(size, 1, 0));
194   }
```



```
196   /*****************************************************************
197    * REST_ScanHyper
198    *
199    * Description:
200    *   This routine will retrieve a hyper from the given string.
201    *
202    * Parameters:
203    *   string (I) - The string to read
204    *   size   (I) - The hyper to read into
205    *
206    * Returns:
207    *   None.
208    *
209    *****************************************************************/

211   void REST_ScanHyper (Str       string,
212                        u_hyper *size)
213 1 {
214 1   /* Call the esl routine to read the hyper */
215 1   string_to_u_hyper (string, size);
216   }
```

```
218   /*****************************************************************
219    * REST_GetGroupString
220    *
221    * Description:
222    *   This routine will return a string representation of the group
223    *   for the given info.
224    *
225    * Parameters:
226    *   info (I) - The item to get the group for.
227    *
228    * Returns:
229    *   A string representation of the group (Copy immediately)
230    *
231    *****************************************************************/

231   Str REST_GetGroupString (RestoreInfoPtr info)
234 1 {
235 1   static Char  returnString[SMALL_STRING_LENGTH];
                                                  /* String to return */
236 1   Str          groupString;

238 1   if (info->restoreObject != NULL)
239 2   {
240 2     groupString = (Str)EDMRST_GetObjectGroupString (GREST_Handle,
241 2                                                     info->restoreObject);

242 2     if (groupString != NULL)
243 3     {
244 3       STR_Cpy (returnString, groupString);
245 2     }
246 2     else
247 3     {
248 3       STR_Cpy (returnString, "");
249 2     }
250 1   }
251 1   else
252 2   {
253 2     STR_Cpy (returnString, "");
254 1   }

256 1   return (returnString);
257   }
```



```
259   /*****************************************************************
260    * REST_GetOwnerString
261    *
262    * Description:
263    *   This routine will return a string representation of the Owner
264    *   for the given info.
265    *
266    * Parameters:
267    *   info (I) - The item to get the Owner for.
268    *
269    * Returns:
270    *   A string representation of the Owner (Copy immediately)
271    *
272    *****************************************************************/

274   Str REST_GetOwnerString (RestoreInfoPtr info)
275 1 {
276 1   static Char  returnString[SMALL_STRING_LENGTH];
                                                  /* String to return */
277 1   Str          ownerString;

279 1   if (info->restoreObject != NULL)
280 2   {
281 2     ownerString = (Str)EDMRST_GetObjectOwnerString (GREST_Handle,
282 2                                                     info->restoreObject);

283 2     if (ownerString != NULL)
284 3     {
285 3       STR_Cpy (returnString, ownerString);
286 2     }
287 2     else
288 3     {
289 3       STR_Cpy (returnString, "");
290 2     }
291 1   }
292 1   else
293 2   {
294 2     STR_Cpy (returnString, "");
295 1   }

297 1   return (returnString);
298   }
```

```
300  /*****************************************************************
301   * REST_GetPermString
302   *
303   * Description:
304   *   This routine will return a string representation of the
                                                          Permissions
305   *   for the given info.
306   *
307   * Parameters:
308   *   info (I) - The item to get the Permissions for.
309   *
310   * Returns:
311   *   A string representation of the Permissions (Copy immediately)
312   *
313   *****************************************************************/
315  Str REST_GetPermString (RestoreInfoPtr info)
316 1{
317 1  static Char  returnString[SMALL_STRING_LENGTH];
                                                  /* String to return */
318 1  Str          permString;
320 1  if (info->restoreObject != NULL)
321 2  {
322 2    permString = GREST_GetPermissionsString (
                                        GREST_Handle, info->restoreObject);
323 2    if (permString != NULL)
324 3    {
325 3      STR_Cpy (returnString, permString);
326 2    }
327 2    else
328 3    {
329 3      STR_Cpy (returnString, "");
330 2    }
331 1  }
332 1  else
333 2  {
334 2    STR_Cpy (returnString, "");
335 1  }
337 1  return (returnString);
338  }
```



```
340  /*****************************************************************
341   * REST_GetTimeDateString
342   *
343   * Description:
344   *   This routine will return a string representation of the time and
                                                                  date
345   *   for the given info.
346   *
347   * Parameters:
348   *   info (I) - The item to get the time and date for.
349   *
350   * Returns:
351   *   A string representation of the time and date (Copy immediately)
352   *
353   *****************************************************************/
355  Str REST_GetTimeDateString (time_t theTime)
356 1{
357 1  static Char  returnString[MEDIUM_STRING_LENGTH];
                                                  /* String to return */
359 1  strftime (returnString,
360 1            MEDIUM_STRING_LENGTH,
361 1            "%m/%d/%y %H:%M",
362 1            localtime (&theTime));
364 1  return (returnString);
365  }
```

```
/*****************************************************************
 * REST_GetSizeString
 *
 * Description:
 *   This routine will return a string representation of the size
 *   for the given info.
 *
 * Parameters:
 *   info (I) - The item to get the size for.
 *
 * Returns:
 *   A string representation of the size (Copy immediately)
 *
 *****************************************************************/

Str REST_GetSizeString (RestoreInfoPtr info)
{
  static Char  returnString[MEDIUM_STRING_LENGTH];
                                                /* String to return */

  u_hyper      size;
                                                /* Size fo the object */


  if ((info->type != REST_Client) &&
      (info->type != REST_WorkItem) &&
      (info->restoreObject != NULL))
  {
    size = EDMRST_GetObjectSize(GREST_Handle, info->restoreObject);
  }
  else
  {
    size.high = 0;
    size.low = 0;
  }

  REST_SPrintHyper (returnString, size);

  return (returnString);
}
```



```
/*****************************************************************
 * REST_GetDateString
 *
 * Description:
 *   This routine will return a string representation of the Date
 *   for the given info.
 *
 * Parameters:
 *   info (I) - The item to get the Date for.
 *
 * Returns:
 *   A string representation of the Date (Copy immediately)
 *
 *****************************************************************/

Str REST_GetDateString (RestoreInfoPtr info)
{
  static Char  returnString[MEDIUM_STRING_LENGTH];
                                                /* String to return */

  time_t       theTime;
                                                /* Time of the object */


  if (info->restoreObject != NULL)
  {
    theTime = EDMRST_GetObjectModDate(
                                        GREST_Handle, info->restoreObject);

    strftime (returnString,
              MEDIUM_STRING_LENGTH,
              "%b %d",
              localtime (&theTime));
  }
  else
  {
    STR_Cpy (returnString, "");
  }
  return (returnString);
}
```

```
/*****************************************************************
 * REST_GetTimeString
 *
 * Description:
 *   This routine will return a string representation of the Time
 *   for the given info.
 *
 * Parameters:
 *   Info (I) - The item to get the Time for.
 *
 * Returns:
 *   A string representation of the Time (Copy immediately)
 *
 *****************************************************************/

Str REST_GetTimeString (RestoreInfoPtr info)
{
  time_t        modificationTime;
                                                /* Time of the object */
  time_t        now;                            /* Current time */
  static Char   returnString[MEDIUM_STRING_LENGTH];
                                                /* String to return */

  if (info->restoreObject != NULL)
  {
    /* Get the modification time */
    modificationTime = EDMRST_GetObjectModDate (GREST_Handle,
                                                info->restoreObject);

    /* If the time is more than 6 months old, show the year only */
    now = time((time_t *)NULL);
    if ((now - modificationTime) > (SECONDS_PER_YEAR / 2))
    {
      strftime (returnString,
                MEDIUM_STRING_LENGTH,
                "%Y",
                localtime (&modificationTime));
    }
    /* Otherwise, show the time */
    else
    {
      strftime (returnString,
                MEDIUM_STRING_LENGTH,
                "%H:%M",
                localtime (&modificationTime));
    }
  }
  else
  {
    STR_Cpy (returnString, "");
  }
  return (returnString);
}
```



```
/*****************************************************************
 * REST_StripDirectoryChars
 *
 * Description:
 *   This routine will strip off the ending directory characters from
 *   the given string;
 *
 * Parameters:
 *   dirString (I) - The string to strip
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_StripDirectoryChars (Str dirString)
{
  BoolEnum      lastCharFound = BOOL_FALSE;
                                            /* Flag if we verified last char */
  Int           lastChar;
                                            /* Current position of last char */

  /*
   * Strip off trailing spaces and directory markers
   */

  lastChar = STR_Len (dirString) - 1;
  while ((!lastCharFound) && (lastChar > 0))
  {
    /* Check if the last character is a blank or directory marker */
    if ((dirString[lastChar] == ' ') ||
        (dirString[lastChar] == '/') ||
        (dirString[lastChar] == '\\'))
    {
      /* Remove the character from the string */
      dirString[lastChar] = '\0';
      lastChar--;
    }
    else
    {
      /* Valid character */
      lastCharFound = BOOL_TRUE;
    }
  }
}
```

```
/*****************************************************************
 * REST_GetFullName
 *
 * Description:
 *   This routine will return a string representation of the FullName
 *   for the given info.
 *
 * Parameters:
 *   info (I) - The item to get the FullName for.
 *
 * Returns:
 *   A string representation of the FullName (Copy immediately)
 *
 *****************************************************************/

Str REST_GetFullName (RestoreInfoPtr info)
{
  static Str  returnString = NULL;          /* The string to return */


  /* Free the old string if necessary */
  if (returnString != NULL)
    GUTIL_Free (returnString);

  /* If this is a client, the full name is the name */
  if (info->type == REST_Client)
  {
    returnString = esl_strdup (info->name);
  }

  /* If this is a restore object,
                          get the full name from the Restore API */
  else if (info ->restoreObject != NULL)
  {
    returnString = esl_strdup (KDMRST_GetObjectFullName (
                                  GREST_Handle, info->restoreObject));
  }

  /* Else, just return the name (this should never happen) */
  else
  {
    returnString = esl_strdup (info->name);
  }

  /* Strip off any trailing directory characters */
  if ((info->type != REST_Client)    &&
      (info->type != REST_WorkItem)  &&
      (info->type != REST_FailedWorkItem))
  {
    REST_StripDirectoryChars (returnString);
  }

  return (returnString);
}
```



```
/*****************************************************************
 * REST_GetNameString
 *
 * Description:
 *   This routine will return a string representation of the Name
 *   for the given info.
 *
 * Parameters:
 *   fileObject (I) - The item to get the Name for.
 *
 * Returns:
 *   A string representation of the Name (Copy immediately)
 *
 *****************************************************************/

Str REST_GetNameString (GPMGR_Context fMgr,
                        GPMGR_Object  fileObject)
{
  RestoreInfoPtr  info; /* Real info for the object */

  info = (RestoreInfoPtr)fileObject;

  return (info->name);
}
```

```
/*****************************************************************
 * REST_IsParentString
 *
 * Description:
 *   This routine will return whether or not the given parent is the
 *   parent of the given full child name.
 *
 * Parameters:
 *   parent     (I) - The parent to check.
 *   childName (I) - The full name of the child looking for.
 *
 * Returns:
 *   BOOL_TRUE  - if it is the parent.
 *   BOOL_FALSE - if it is not the parent.
 *
 *****************************************************************/

BoolEnum  REST_IsParentString (RestoreInfoPtr parent,
                               Str            childName)
{
  Str       fullParentName; /* Full path for parent */
  Int       length;
                            /* Length of the full path name for parent */
  BoolEnum retVal;          /* Value to return */

  /* Get the full path name for the parent object */
  fullParentName = esl_strdup (REST_GetFullName(parent));
  length = STR_Len(fullParentName);

  /* If the parent name ends in a directory marker,
                                        compare only before it */
  if ((fullParentName[length-1] == '\\') ||
      (fullParentName[length-1] == '/'))
  {
    /* Decrease our comparison by a character */
    length--;
  }

  /*
   * Check that they match up to the end of the parent name
   * and then the next child character is the dir symbol ('/')
   * or the end of the string (files are children of themselves).
   */

  if (STR_NCmp (fullParentName, childName, length) == CMP_EQUAL)
  {
    retVal = ((childName[length] == '\0') ||
              (childName[length] == '\\') ||
              (childName[length] == '/'));
  }
  else
  {
    retVal = BOOL_FALSE;
  }

  /* Now free the full name */
  GUTIL_Free (fullParentName);

  /* return */
  return (retVal);
}
```



```
/*****************************************************************
 * REST_GetStatusIcon
 *
 * Description:
 *   This routine will return the status overlay icon for the given
 *   restore object status.
 *
 * Parameters:
 *   status (I) - The status of the object.
 *
 * Returns:
 *   IconPtr - The icon to display as the overlay icon, possibly NULL
 *
 *****************************************************************/

IconPtr  REST_GetStatusIcon (BackupStatus status)
{
  IconPtr returnIcon;

  /* If expired, return the expired icon */
  if (status == Backup_Expired)
  {
    returnIcon = REST_ExpiredIcon;
  }

  /* Else If expired children, return the missing children icon */
  else if (status == Backup_Child_Without_Data)
  {
    returnIcon = REST_MissingChildrenIcon;
  }

  /* Else If the object status is bad, return the bad icon */
  else if (status == Backup_Bad)
  {
    returnIcon = REST_BadIcon;
  }

  /* Else return No icon */
  else
  {
    returnIcon = NULL;
  }

  return (returnIcon);
}
```

```
718  /*****************************************************************
719   * REST_GetIcon
720   *
721   * Description:
722   *   This routine will return the icon to display for the given object.
723   *   This routine is generally called from the file manager.
724   *
725   * Parameters:
726   *   fileObject (I) - The object
727   *   isOpen     (I) - Flag if the object is "open"
728   *
729   * Returns:
730   *   The icon to display (can be NULL).
731   *
732   *****************************************************************/

734  IconPtr REST_GetIcon  (GFMGR_Context fMgr,
735                         GFMGR_Object  fileObject,
736                         BoolEnum      isOpen)
737  1 {
738  1   RestoreInfoPtr  info;          /* The real info for the object */
739  1   IconPtr         returnIcon;    /* The icon to display */
740
741  1   /* Cast back to the real object */
742  1   info = (RestoreInfoPtr)fileObject;
743
744  1   /* Get the status icon */
745  1   returnIcon = REST_GetStatusIcon (info->status);
746
747  1   /* If status does not need to be displayed,
                                              display the object icon */
748  1   if (returnIcon == NULL)
749  2   {
750  2     /* If this object is open and it is a directory,
                                              return the dir open icon */
751  2     if ((isOpen) && (info->type == REST_Directory))
752  2       returnIcon = REST_DirOpenIcon;
753
754  2     /* Else return the current icon for the object */
755  2     else
756  2       returnIcon = info->icon;
757  1   }
758
759  1   /* Return the icon */
760  1   return (returnIcon);
761  }
```



```
763  /*****************************************************************
764   * REST_GetIcon2
765   *
766   * Description:
767   *   This routine will return the icon to display for the given object.
768   *   This routine is generally called from the file manager.
769   *
770   * Parameters:
771   *   fileObject (I) - The object
772   *   isOpen     (I) - Flag if the object is "open"
773   *
774   * Returns:
775   *   The icon to display (can be NULL).
776   *
777   *****************************************************************/

779  IconPtr REST_GetIcon2 (GFMGR_Context fMgr,
780                         GFMGR_Object  fileObject,
781                         BoolEnum      isOpen)
782  1 {
783  1   RestoreInfoPtr  info;          /* The real info for the object */
784  1   IconPtr         tempIcon;      /* Temporary Icon */
785  1   IconPtr         returnIcon;    /* The icon to display */
786
787  1   /* Cast back to the real object */
788  1   info = (RestoreInfoPtr)fileObject;
789
790  1   tempIcon = REST_GetStatusIcon (info->status);
791
792  1   /* If status needed to be displayed,
                                              display the object icon second */
793  1   if (tempIcon != NULL)
794  2   {
795  2     /* If this object is open and it is a directory,
                                              return the dir open icon */
796  2     if ((isOpen) && (info->type == REST_Directory))
797  2       returnIcon = REST_DirOpenIcon;
798
799  2     /* Else return the current icon for the object */
800  2     else
801  2       returnIcon = info->icon;
802  1   }
803  1   else
804  2   {
805  2     returnIcon = NULL;
806  1   }
807
808  1   /* Return the icon */
809  1   return (returnIcon);
810  }
```

```
812  /*****************************************************************
813   * REST_GetOverlayIcon
814   *
815   * Description:
816   *   This routine will return the overlay icon to display for the
                                                          given object.
817   *   This routine is generally called from the file manager.
818   *
819   * Parameters:
820   *   fileObject (I) - The object
821   *
822   * Returns:
823   *   The overlay icon to display (can be NULL).
824   *
825   *****************************************************************/

827  IconPtr REST_GetOverlayIcon  (GFMGR_Context fMgr,
828                                GFMGR_Object  fileObject)
829  {
830    return (NULL);
831  }
```



```
833  /*****************************************************************
834   * REST_GetOverlayIcon2
835   *
836   * Description:
837   *   This routine will return the overlay icon to display for the
                                                          given object.
838   *   This routine is generally called from the file manager.
839   *
840   * Parameters:
841   *   fileObject (I) - The object
842   *
843   * Returns:
844   *   The overlay icon to display (can be NULL).
845   *
846   *****************************************************************/

848  IconPtr REST_GetOverlayIcon2 (GFMGR_Context fMgr,
849                                GFMGR_Object  fileObject)
850  {
851    return (NULL);
852  }
```

```
/*****************************************************************
 * REST_DisposeInfo
 *
 * Description:
 *   This routine will free the info memory associated with the given
 *                                                          object.
 *
 * Parameters:
 *   info        (I) - The item to dispose of.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_DisposeInfo (RestoreInfoPtr info)
{
  /* Sanity check, make sure no bonehead called us with a NULL info */
  if (info != NULL)
  {

    /* Free up the name for this obect */
    GUTIL_Free (info->name);

    /* Free up the error string if it existed */
    if (info->errorString != NULL)
      GUTIL_Free (info->errorString);

    /* Free up the object */
    GUTIL_Free (info);
  }
}
```



```
/*****************************************************************
 * REST_FreeInfo
 *
 * Description:
 *   This routine will free all memory associated with the given
                                                          object
 *   and all of the object's children.
 *
 * Parameters:
 *   info        (I) - The item to free.
 *
 * Returns:
 *   None.
 *
 *****************************************************************/

void REST_FreeInfo (RestoreInfoPtr info)
{
  RestoreInfoPtr  childInfo;  /* Child info to free */
  RestoreInfoPtr  nextChild;  /* Pointer to next child */

  /* Make sure there is info to free */
  if (info != NULL)
  {

    /* Free up any children of this object */
    childInfo = info->children;
    while (childInfo != NULL)
    {
      nextChild = childInfo->next;
      REST_FreeInfo (childInfo);
      childInfo = nextChild;
    }

    /* Free up the associated restorable object if it exists */
    if (info->restoreObject != NULL)
    {
      EDMRST_FreeRestorableObjects (
                                    GREST_Handle, &info->restoreObject, 1);
    }

    /* Free up the memory for this obect */
    REST_DisposeInfo (info);
  }
}
```

```
/*****************************************************************
 * REST_IsLessThan
 *
 * Description:
 *    This routine determins if an object is "less than" another object
 *    based on current sort criteria.
 *
 * Parameters:
 *    item1 (I) - The item in question
 *    item2 (I) - The item to compare to
 *
 * Returns:
 *    BOOL_TRUE  - if item1 < item2
 *    BOOL_FALSE - if item1 >= item2
 *
 *****************************************************************/

BoolEnum REST_IsLessThan (RestoreInfoPtr item1,
                          RestoreInfoPtr item2)
{
  Char            string1[GMAX_OBJECT_LENGTH]; /* String for item1 */
  Char            string2[GMAX_OBJECT_LENGTH]; /* String for item2 */
  u_hyper         size1;                       /* Size for item1 */
  u_hyper         size2;                       /* Size for item2 */
  CmpEnum         cmpValue;
                                   /* Value returned by comparison */
  BoolEnum        returnValue;                 /* Value to return */
  REST_SortType   currentSort;
                                   /* Current type of sort to use */

  /* Based on the current sort criteria, compare the two items */
  currentSort = REST_GetSort ();
  switch (currentSort)
  {

    /* If sorting by name, compare the names */
    case REST_ByName:
      returnValue = (STR_Cmp (item1->name, item2->name) == CMP_UNDER);
      break;

    /* If sorting by type, compare the types */
    case REST_ByType:
      /* If they are the same type, sort by name */
      if (item1->type == item2->type)
        returnValue = (STR_Cmp (
                                item1->name, item2->name) == CMP_UNDER);
      else
        returnValue = (item1->type < item2->type);
      break;

    /* If sorting by date, compare the dates */
    case REST_ByDate:
      /* Date works backwards, sort by latest time first */
      returnValue = (EDMRST_GetObjectModDate (GREST_Handle,
                                              item1->restoreObject) >
                     EDMRST_GetObjectModDate (GREST_Handle,
                                              item2->restoreObject));
      break;

    /* If sorting by size, compare the sizes */
    case REST_BySize:
      /* Size works backwards, sort by largest size first */
      size1 = EDMRST_GetObjectSize(GREST_Handle, item1->restoreObject);
      size2 = EDMRST_GetObjectSize(GREST_Handle, item2->restoreObject);
      if (size1.high > size2.high)
        returnValue = BOOL_TRUE;
      else if ((size1.high == size2.high) && (size1.low > size2.low))
        returnValue = BOOL_TRUE;
      else if ((size1.high == size2.high) && (size1.low == size2.low))
        returnValue = (STR_Cmp (
                                item1->name, item2->name) == CMP_UNDER);
      else
        returnValue = BOOL_FALSE;
      break;

    /* If sorting by owner, compare the owners */
    case REST_ByOwner:
      STR_Cpy (string1, REST_GetOwnerString(item1));
      STR_Cpy (string2, REST_GetOwnerString(item2));
      cmpValue = STR_Cmp (string1, string2);
      if (cmpValue == CMP_UNDER)
        returnValue = BOOL_TRUE;
      else if (cmpValue == CMP_EQUAL)
        returnValue = (STR_Cmp (
                                item1->name, item2->name) == CMP_UNDER);
      else
        returnValue = BOOL_FALSE;
      break;

    default:
      /* Hmmmm... Oh well, sort by name. */
      returnValue = (STR_Cmp (item1->name, item2->name) == CMP_UNDER);
  }

  /* Return the value determined */
  return (returnValue);
}
```

```
1024  /*****************************************************************
1025   * REST_IsBadObject
1026   *
1027   * Description:
1028   *   This routine will determine if the object status is bad.
1029   *
1030   * Parameters:
1031   *   info   (I) - The object to check the status of.
1032   *
1033   * Returns:
1034   *   BOOL_TRUE  - If the object status is bad
1035   *   BOOL_FALSE - otherwise
1036   *
1037   *****************************************************************/

1039  BoolEnum REST_IsBadObject (RestoreInfoPtr info)
1040  1 {
1041  1   BoolEnum returnValue;  /* Value to return */


1043  1   /* If this is a Restore API object and it has a bad status */
1044  1   if (info->status == Backup_Bad)
1045  2   {
1046  2     /* This is a bad, bad, object */
1047  2     returnValue = BOOL_TRUE;
1048  1   }
1049  1   else
1050  2   {
1051  2     /* This object has been very good */
1052  2     returnValue = BOOL_FALSE;
1053  1   }

1055  1   /* Return whether or not the object is bad */
1056  1   return (returnValue);
1057    }
```



```
1059  /*****************************************************************
1060   * REST_HasChildren
1061   *
1062   * Description:
1063   *   This routine is provided for the file manager.  It determines
1064   *   if the given object can have children.
1065   *
1066   * Parameters:
1067   *   fileObject (I) - Object to check for children.
1068   *
1069   * Returns:
1070   *   None.
1071   *
1072   *****************************************************************/

1074  BoolEnum REST_HasChildren (GFMGR_Context fMgr,
1075                             GFMGR_Object  fileObject)
1076  1 {
1077  1   RestoreInfoPtr    info;
                                  /* Real representation of the object */
1078  1   BoolEnum          returnValue; /* Value to return */

1080  1   /* Got the real data type for the object */
1081  1   info = (RestoreInfoPtr)fileObject;

1083  1   /* If this is a directory, client, or a work-item,
                                                    it has children */
1084  1   if ((info->type == REST_Directory)  ||
1085  1       (info->type == REST_Client)     ||
1086  1       (info->type == REST_WorkItem)   ||
1087  1       (info->type == REST_FailedWorkItem))
1088  2   {
1089  2     returnValue = BOOL_TRUE;
1090  1   }

1092  1   /* Otherwise it has no children */
1093  1   else
1094  2   {
1095  2     returnValue = BOOL_FALSE;
1096  1   }

1098  1   /* Return whether or not it has children */
1099  1   return (returnValue);
1100    }
```

```
1102  /*****************************************************************
1103   * REST_StandardizePath
1104   *
1105   * Description:
1106   *    This routine is provided to convert the given path into
1107   *    standard UNIX format. i.e., C:\ABC will be modified as /C/ABC
1108   *    All backslashes will be made forwardslashes if a colon appears
1109   *    after the first Alphabet.
1110   *
1111   * Parameters:
1112   *    Str : The data in this string is modified.
1113   *
1114   * Returns:
1115   *    BOOL_TRUE
1116   *
1117   *****************************************************************/

1119  BoolEnum REST_StandardizePath(Str pathname)
1120  {
1121     int i;
1122     int pathlength = strlen(pathname);

1124     if(pathname == NULL || pathlength < 2 )
1125        return BOOL_FALSE;

1127     /* NT style paths have first character as a drive name, i.e.,
                                                          a single
1128        character followed by a colon */

1130     if( ':' == pathname[1]  &&
1131           ( toupper(pathname[0]) >= 'A' && toupper(
                                              pathname[0]) <= 'Z' ))
1132     {
1133        pathname[1] = pathname[0];
1134        pathname[0] = '/';
1135     }

1137     /* If it is NT style path,
                                then the backslashes should also be converted
1138      * into forward slashes.
1139      */
1140     for(i=2; i<pathlength; i++ )
1141     {
1142        if( '\\' == pathname[i] )     /* need another back slash to hide
                                                                 it */
1143           pathname[i] = '/';
1144     }
1145     return BOOL_TRUE;
1146  }
```

```
/*****************************************************************
 * restStartMgr.c
 *
 *
 * Copyright 1996 by Epoch Systems, Inc.
 *
 *
 * Mission Statement:
 *    This file contains the functions necessary for starting the
                                                          actual
 *    the EDM Restoral.
 *
 * Required includes:
 *    None
 *
 * Compile-Time Options:
 *    N/A
 *
 *
 * RCS Information:
 *       $RCSfile$
 *       $Revision$
 *       $Date$
 *****************************************************************/

#define  ERR_LIB  RESTORE

#include <esl/c_portable.h>
#include <esl/ep_xopen.h>
#include <util/esl_core.h>
#include <eerrno/e_errno.h>
#include <util/esl_string.h>

#include <stdlib.h>

/* WARNING: UNIX DEPENDENCY!!! Used for polling sockets */
#include <stropts.h>
#include <poll.h>
#include <unistd.h>

#include <libgen.h>
#include <time.h>

#include <apppub.h>
#include <eventpub.h>
#include <rlibpub.h>
#include <gwpub.h>
#include <respub.h>
#include <lboxpub.h>
#include <tbutpub.h>
#include <tedpub.h>
#include <winpub.h>
#include <strlpub.h>
#include <drawpub.h>
#include <dsplypub.h>

#include "eerrno/e_errno.h"
#include "fbrowser/epcomm_api.h"
#include "util/hyper.h"

#include <restore/restore_api.h>
#include "restore/restStartMgr.h"
#include "restoreP.h"
#include "restUtils.h"
#include "restSearch.h"
#include "restSelMgr.h"
```



```
#include "restCBMgr.h"
#include "restDest.h"
#include "restProgress.h"
#include "restQuestion.h"
#include "gutil/timeUtils.h"
#include "gutil/winUtils.h"
#include "gutil/guiDefines.h"
#include "gutil/guiUtils.h"
#include "gutil/iconDefs.h"
#include "gutil/iconUtils.h"
#include "gutil/alertMgr.h"

/************************
 * Local Data Structures *
 ************************/

typedef struct _REST_TimerRec
{
  int     startFd;          /* The File descriptor to get data from */
  void    (*auxprocRead)();
                            /* The routine to call when data is received */
  void   *data;             /* Data to pass on to the read routine */
} REST_TimerRec, *REST_TimerPtr;

/*****************************************************************
 * REST_VerifyMedia
 *
 * Description:
 *    This routine will verify the media for the current restore
 *
 * Parameters:
 *    None.
 *
 * Returns:
 *    BOOL_TRUE  - If it is OK to proceed with restore
 *    BOOL_FALSE - If there is offline/offsite media and the user
                                                        cancelled
 *
 *****************************************************************/

static BoolEnum REST_VerifyMedia ( void )
{
  GREST_MediaObject nextMedia;
                                          /* Next media object in list */
  BoolEnum          OKtoRestore = BOOL_TRUE;
                                          /* Flag if OK to proceed */
  BoolEnum          errorDisplayed = BOOL_FALSE;
                                          /* Flag if we already asked */


  /*
   * Loop through the current media list,
   * If any one is not online,
                                  ask the user if he/she wants to proceed.
   */

  LBOX_GoColiRow (REST_RestoreWin->MediaListBox, 1);
  nextMedia = (GREST_MediaObject) LBOX_CurGetClientData (
                                          REST_RestoreWin->MediaListBox);
  while ((nextMedia != NULL) && (!errorDisplayed))
  {
    /* IF the media is offline or offsite, display a warning */
    if (EDMREST_GetMediaStatus (
                                  GREST_Handle, nextMedia) != Media_Online)
    {

      /* Flag that the error has been displayed */
```

```
124 3       errorDisplayed = BOOL_TRUE;

126 3       /* Ask the user if he/she wants to continue anyways */
127 3       if (GALERT_DisplayQuestion ((WinPtr)REST_RestoreWin,
128 3                                   REST_GetErrorString(
                                            REST_OFFLINE_MEDIA_TITLE),
                                    GICON_GetWarning(),
129 3                               REST_GetErrorString(
130 3                                       REST_OFFLINE_MEDIA_ERROR),
131 3                               BOOL_FALSE) != GALERT_Affirmative)

133 4       {
134 4         /* The user cancelled the restore */
135 4         OKtoRestore = BOOL_FALSE;
136 3       }
137 2     }
138 2     LBOX_GoDown (REST_RestoreWin->MediaListBox);
139 2     nextMedia = (GREST_MediaObject) LBOX_CurGetClientData (
                                        REST_RestoreWin->MediaListBox);
140 1   }

142 1   /* Return whether or not it is OK to proceed with the restore */
143 1   return (OKtoRestore);
144   }
```



```
146   /*****************************************************************
147    * REST_StartRestoral
148    *
149    * Description:
150    *   This routine start the restoral of the currently marked objects.
151    *
152    * Parameters:
153    *   None.
154    *
155    * Returns:
156    *   None.
157    *
158    *****************************************************************/

160   void REST_StartRestoral (void)
161 1 {
162 1   BoolEnum            inPlace;                /* Flag if this is inplace */
163 1   Char                hostName[MAX_CLIENT_NAME_LENGTH];
                                                    /* Host to restore to */
164 1   Char                destHostName[MAX_CLIENT_NAME_LENGTH];
                                                    /* Host to restore to */
165 1   Char                dirName[MAX_STRING_LENGTH];
                                                    /* Directory to restore to */
166 1   RestoreInfoPtr      tmpInfo;                /* Pointer to walk list */
167 1   OverwritePolicy     policy;
                                                    /* Overwrite policy */
168 1   RestoreTransport    transport;
                                                    /* Data Transport type */
169 1   eerrno_ty           eerrno;                 /* Error Status */

171 1   /* Do nothing if we're searching, or a restore is in progress */
172 1   if (REST_SearchInProgress() ||
173 1       REST_RestoreInProgress() ||
174 1       (currentWorkItemInfo == NULL))
175 2   {
176 2     return;
177 1   }

179 1   /* Verify that the media is online or the user doesn't care */
180 1   if (!REST_VerifyMedia ())
181 2   {
182 2     return;
183 1   }

185 1   /*
186 1    * Find the client object for this work-item
187 1    * Default to in place host and directory
188 1    */

190 1   /* Use the current info */
191 1   tmpInfo = currentWorkItemInfo;

193 1   /* Find the client for this restoral */
194 1   while ((tmpInfo != NULL) && (tmpInfo->type != REST_Client))
195 2   {
196 2     tmpInfo = tmpInfo->parent;
197 1   }

199 1   /* If we found the client get its name */
200 1   if (tmpInfo != NULL)
201 2   {
202 2     STR_Cpy (hostName, tmpInfo->name);
203 1   }
```

```
205 1     /* Else we have no host ?? */
206 1     else
207 2     {
208 2       STR_Cpy (hostName, "");
209 1     }
210 1     /* Default the directory name to the root directory */
211 1     STR_Cpy (dirName, "/");
212 1
213 1     if (REST_GetDestinationInfo ((WinPtr)REST_RestoreWin,
214 1                                  &inPlace,
215 1                                  destHostName,
216 1                                  dirName,
217 1                                  &policy,
218 1                                  &transport))
219 2     {
220 2       /* Submit the restoral */
221 2       if ((eerrno = EDMRST_Submit (GREST_Handle,
222 2                                    destHostName,
223 2                                    policy,
224 2                                    inPlace,
225 2                                    dirName,
226 2                                    transport,
227 2                                    0,
228 2                                    NULL)) != 0)
229 2       {
230 3         REST_DisplayErrnoMessage ((WinPtr)REST_RestoreWin,
231 3                                   NULL,
232 3                                   REST_GetErrorString (
233 3                                                        REST_START_ERROR),
234 3                                   eerrno);
235 2       }
236 2       else
237 1       {
238 1         REST_StartProgress ();
239 2       }
240 1     }
241   }
```